8th European Congress

EMBEDDED REAL TIME SOFTWARE AND SYSTEMS

ERTS² 2016

27-29 JANUARY 2016 / TOULOUSE, FRANCE

CENTRE DE CONGRÈS PIERRE BAUDIS

PROCEEDINGS

Contents

Program Committee

Canals Agusti	CS
Albert Aribaud	3ADEV SAS
Eric Armengaud	AVL List GmbH
Philippe Baufreton	Sagem
Scott Beecher	Pratt and Whitney - United Technologies Corp.
Jean-Louis Bergerand	Schneider-Electric
Jean-Paul Blanquart	Airbus Defence and Space
Siegfried Bocionek	Siemens AG
Jean-Louis Boulanger	CERTIFER
Marc Boyer	ONERA
Jens Braband	Siemens AG
Duncan Brown	Controls and Data Services
Lionel Burgaud	Assystem
Denis Claraz	Continental Automotive SAS
Darren Cofer	RockwellCollins
Cyrille Comar	Adacore
Eric Conquet	ESA
Patrick Cormery	Astrium Space Transportation
Philippe Cuenot	Continental
Jean-Charles Damery	CNES
Werner Damm	University of Oldenburg
Herve Delseny	Airbus
Jean-Luc Dormoy	EDF Group
Benoit Dupont De Dinechin	Kalray
Yannick Déléris	Airbus
Faure Eric	ASTC Design Partners
Francesco Flammini	Ansaldo STS Italy, Univ. ”Federico II” of Naples
Pedro Gil Vicente	Universidad Politécnica de Valencia
Olivier Guetta	Renault
Kelly Hayhurst	NASA
Eric Jenn	IRT Saint-Exupery
Ahmed Jerraya	CEA
Chris Johnson	University of Glasgow
Mohamed Kaaniche	LAAS-CNRS
Tim Kelly	University of York
Maged Khalil	Fortiss
Uwe Kuehne	EADS - Cassidian Electronics
Chidung Lac	Orange
Gerard Ladier	Aerospace Valley
Gilles Le Calvez	VALEO
Emmanuel Ledinot	Dassault Aviation
Andreas Luedtke	OFFIS, Institute for Information Technology
Henrik Lönn	Volvo Technology

Additional Reviewers

Session 1

Avionic Certification

Wednesday 27th, 11:00 – Auditorium St Exupery

Data Flow Model Coverage Analysis: Principles and Practice

Authors: Jean-Louis CAMUS, Carole Haudebourg (Esterel Technologies), Marc Schlickling (Konzept Informationssysteme GmbH), Jörg Barrho (MTU Friedrichshafen GmbH).

Topic Processes, methods and tools

Abstract

Safety critical software requires rigorous processes in order to achieve a high degree of integrity. These processes include so-called “verification of verification”. In the case of Model Based Development and Verification, DO-178C/DO-331 requires model coverage analysis. This paper reminds the objectives of model coverage analysis and the difference with structural code coverage analysis. It proposes coverage criteria suited to data flow models. These criteria are a generalization of the functional masking effect and also take into account modularity and in-lining of the model operators. It presents a tool supporting model coverage analysis according to these criteria. It concludes with industry field experience and future extensions.

Keywords Coverage, Model, Data flow, Tool, Qualification, Safety, Embedded, Software, DO-178C/ED-12C, DO-331, DO-330, EN 50128, IEC 61508, IEC 60880, ISO 26262.

1 Regulatory Context

Safety critical software requires a high degree of rigor in its development and verification processes. These processes are regulated by standards such as DO-178C/ED-12C [1] for airborne software, EN 50128 [2] for railway equipment, IEC 61508 [3] for industry, ISO 26262 [4] for automotive, ECSS (in particular Q80, E40) [5] for European space and IEC 60880 [6] for the nuclear industry. These standards are based on process assurance, with a strong emphasis on verification activities: reviews, analyzes, and testing.

These processes include so-called “verification of verification”. In the case of Model Based Development and Verification, DO-331 [7] requires model coverage analysis of so-called “design models”, i.e. models describing Low-Level Requirements. This paper reminds the objectives of model coverage analysis and the difference with structural code coverage analysis.

1.1 Role of Testing and Coverage Analysis in the Safety Standards

In addition to review and analysis, safety standards require dynamic verification. Simulation can be used as a means for verifying compliance of models to their higher-level requirements and algorithm accuracy. Testing is used for verification of the executable object code.

Safety standards require various types of coverage analysis. For instance DO-178C requires the following categories of coverage analysis:

- High-Level Requirements Coverage analysis (6.4.4.1), with full coverage required by table A-7 objective 3.
- Low-Level Requirements Coverage analysis (6.4.4.1), with full coverage required by table A-7 objective 4. In case of model-based development, the Low-Level Requirements are contained in a model and LLR coverage is model coverage (DO-331 MB 6.4.7).
- Structural [Code] Coverage analysis (6.4.4.2), with full coverage required by table A-7 objectives 5, 6, 7 and 8.

Several aspects of coverage analysis have to be emphasized:

1) Coverage analysis in general (requirements coverage and structural coverage) is not a goal per se, it is a means for verifying that verification activities are thorough and complete (“verification of verification”).

2) One should apply requirements-based testing assessed with structural coverage analysis, and not structural testing. As explained by DO-248C [8] FAQ#44, "*Whereas structural testing is the process of exercising software with test scenarios written from the Source Code, DO-178C/DO-278A section 6.4.4.2 explains that structural coverage analysis determines which code structure, including interfaces between components, was not exercised by the requirements-based test procedures…. Since the starting point for developing structural test cases is the code itself, there is no way of finding requirements (high-level, low-level, or derived) not implemented in the code through structural tests. It is a natural tendency to consider outputs of the actual code (which is de facto the reference for structural testing) as the expected results. This bias is avoided when expected outputs of a tested piece of code are determined by analysis of the requirements.* "

1.2 Objectives of Model Coverage Analysis

1.2.1 Objectives of Model Coverage Analysis

As stated in DO-331 6.7, model coverage analysis determines which requirements expressed by the model were not exercised by verification based on the requirements from which the model was developed. Model coverage analysis is a means to assess thoroughness of the model verification activities. Model coverage gaps may reveal:

a) Shortcomings in requirements-based verification cases or procedures
b) Inadequacies or shortcomings in requirements from which the Model was developed
c) Derived requirements expressed by the Model
d) Deactivated functionality expressed by the Model
e) Unintended functionality expressed by a Model

1.2.2 Difference with Structural Coverage Analysis

Model coverage analysis should not be confounded with structural [code] coverage analysis. Model coverage analysis concerns the Low-Level Requirements expressed by a model, and is at a higher level of abstraction than structural coverage analysis. Model coverage analysis concerns the functional aspects of the model regardless of the code that implements this model, which structure is highly dependent on the coding technique. Also, the code coverage criteria that are required by standards and supported by tools are limited to control structures and Boolean expressions: they do not address very important functional aspects such as the definition and use of data.

1.2.3 Is Model Coverage Analysis a New Burden?

Model coverage analysis is not an additional burden introduced by DO-331, it is just the appropriate term for LLR coverage analysis in case of model-based development and verification (MBDV). LLR coverage analysis has always been required by DO-178B, whether one uses traditional development or MBDV. For years, a number of applicants using MBDV used to assess coverage of their LLRs by manual means such as Excel tables or colored pencil on model diagrams, or using specific tools such as pioneering versions of SCADE MTC or Simulink.

2 Challenge of Defining Appropriate Model Coverage Criteria

2.1 The Scade Language and the SCADE Suite Environment

Let us introduce the context, which is the Scade language and its supporting environment. The Scade language [9] is a synchronous data flow language [10] for reactive software. Its formal semantic foundation is the Lustre language [11], extended with state machines and iterators [12]. Scade is fundamentally a **declarative** language, which is not based on the notion of execution. A Scade model is a set of non-ordered **equations** combining flows/variables subject to formal clocks (i.e. conditions under which are defined). It is not a sequence of executable statements and it is the job of the code generator to transform purely declarative sets of equations into imperative statements.

The Scade language supports a free combination of notations that are familiar to control engineers:

- Block diagrams to specify the control algorithms (control laws, filters)
- State machines to specify modes and transitions in an application (e.g., taking off, landing) with parallelism, hierarchy and preemption
- Flowchart-like constructs (called clocked blocks)

The basic building block in Scade is called an operator. An operator is either a pre-defined operator (e.g., +, delay) or a user-defined operator that is built with predefined or user-defined operators. It is thus possible to build structured complex applications with high modularity.

Elemnt	**Textual Form**	**Graphical Form**

Formal interface	`node IntegrFwd(U: real ; hidden TimeCycle: real)`
	`        returns (Y: real) ;`	

Local variables declarations	`var`
	`delta : real ;`
	`last_y : real;`	

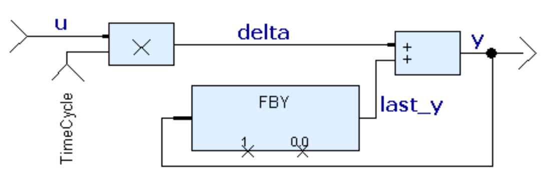
Equations	`delta = u * TimeCycle ;`
	`y = delta + last_y ;`
	`last_y = fby(y , 1 , 0.0) ;`
		 u
		delta
		y
		+
		+
		FBY
		last_y
		TimeCycle
		1
		0.0

Figure 1 Integrator in Graphical Scade Notation

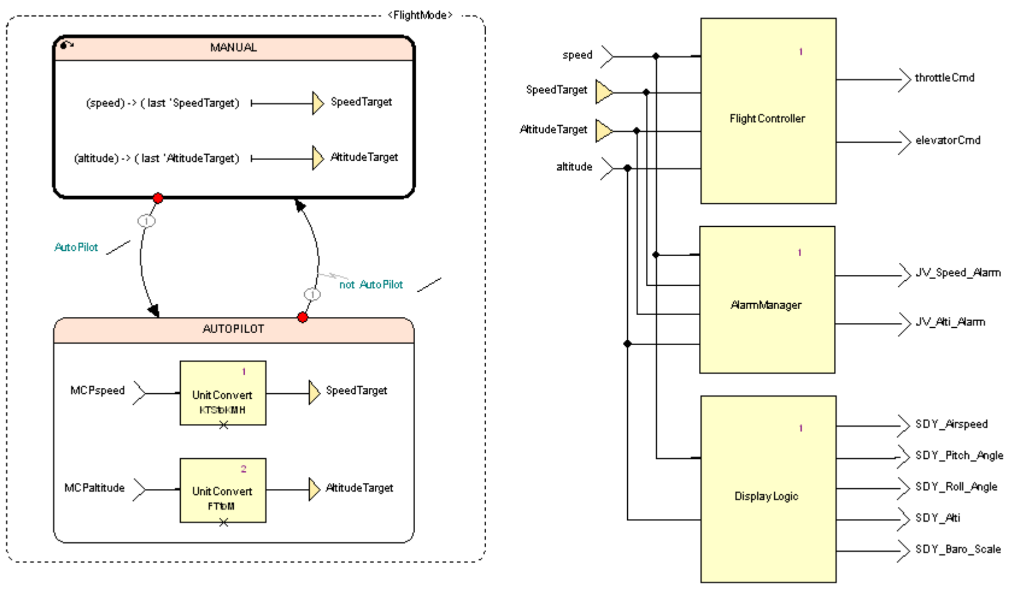


Figure 2: State Machine Combined with Data Flow

SCADE Suite (Safety Critical Application Development Environment) support the development and verification of safety critical software based on the Scade language with:

- Structured graphical editing
- Semantic checks
- Simulation and model coverage analysis
- Formal verification
- Automated code generation qualified for DO-178 level A and IEC 61508/EN 50128 SIL 3-4
- Automated test execution

2.2 Is there a Standard Definition of Model Coverage Criteria?

When writing DO-331, the SC205 SG-4 (subgroup defining the Model Based Development and Verification supplement) has taken into account the fact that there is a large diversity of modeling notations that differ significantly regarding:

- Degree of formality: modeling notations range from non-formal (e.g. UML) to formal (e.g. Scade, SDL).
- They may be based on various concepts and representations such as data flow, state machines, sequence charts.
- They may be based on a synchronous basis (e.g. Scade) and/or an asynchronous basis (e.g. SDL, UML).

And there may also be other notations in the future that are currently unknown.

The committee agreed that it was not possible to impose specific model coverage criteria in DO-331, and the following approach was retained:

1) DO-331 provides general principles for model coverage criteria (e.g. coverage of transitions in state machines) and requires model coverage criteria to be defined in each project's software verification plan.
2) Each applicant project defines in its SVP the model coverage criteria that it will apply.

Thus, it was necessary to define relevant model coverage criteria for the Scade language.

2.3 Are Structural Code Coverage Criteria Suited to Data Flow Model Coverage?

Analysis of classical imperative code has been investigated and applied for decades. This type of analysis is supported by a number of tools. The most commonly used code coverage criteria are Statement Coverage, various forms of Branch Coverage, Decision Coverage (DC), Modified Decision Coverage (MC/DC) [13]. Note that statement coverage and branch coverage address the control structure, whilst DC and MC/DC are actually specific data flow criteria for Boolean expressions (a decision is NOT a branch as recalled by [14].

The first natural approach is to consider traditional structural coverage criteria as candidates for the basis of model coverage criteria. But this approach would not work with Scade for the following reasons:

- It would be highly implementation-dependent
- It would not capture the essence of the functional flows expressed at the model level
- The traditional criteria are based on execution concepts for imperative languages, whilst Scade is a declarative language, not based on the notion of execution (it is the job of the code generator to transform purely declarative sets of equations into imperative statements).

3 Data Flow Coverage Criteria for the Scade Language

3.1 Principles of the Model Coverage Criteria Defined for the Scade Language

The principles of the proposed model coverage criteria are based on two related considerations:

- Good testing practice: an experienced tester develops requirement-based tests checking that the effect of input data values on observed data complies with the requirements. In order to be under appropriate controllability/observability situations, the tester sets the software in conditions where this influence occurs for each concerned data of interest. The proposed model coverage criteria assess that these conditions are met during testing.
- Theoretical basis: a synthesis of *Definition-Use* analysis is provided in [15]. More specifically, [16] proposes coverage criteria for the Lustre language (on which Scade is based). In this paper, we have chosen a definition sharing with it some aspects of path activity. In addition it consistently takes into account masking MC/DC, Arithmetic, Structured data handling, Iterators, Reset, Clocked blocks and State machines.

A data path from one point of the data flow graph to another one is active when its Activation Condition (AC) is true. For instance on Figure 3, B influences Z when C2 is true and C1 is false.

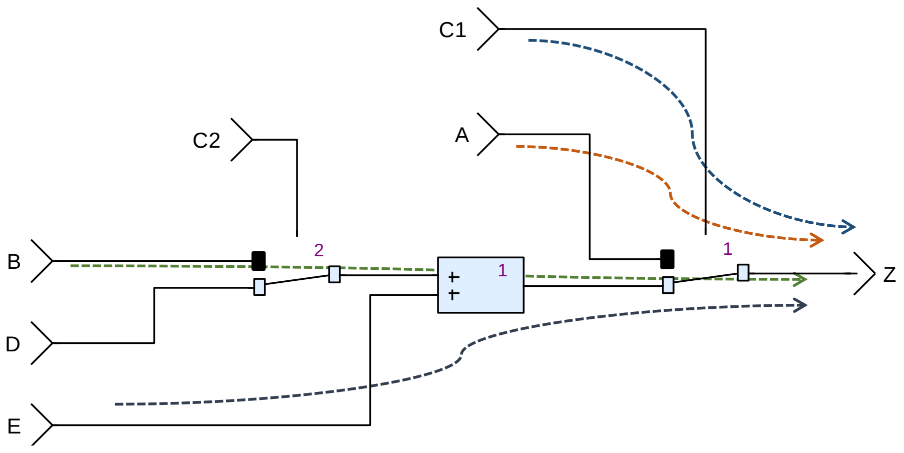


Figure 3: Example 1 Data Flow Activity and Influence

Several coverage criteria all based on this activity concept are defined, as summarized in Table 1.

Table 1: Coverage Criteria Summary

Coverage Criterion	Synopsis
Basic Coverage (BC)	Every element has been on an active path.
Decision Coverage (DC)	Every Boolean expression outcome has toggled while on an active path.
Modified Condition /Decision Coverage (MC/DC)	Every condition of every Boolean expression's has toggled while on an active path (shows independent effect of the conditions).
Control coupling	Every state/transition, clocked block, activate branch has been active
Data coupling	Every input flow to every user operator instance has changed value.
Operator feature	Every characteristic feature (e.g. saturation, reset) has been active.

In order to match industrial practice regarding code coverage analysis, the current approach considers the flows occurring in the current instant (i.e. memories are part of the inputs/outputs).

3.2 Application of the Influence Principle to Logic and State Machines

The principles presented above are very general. In the case of Boolean expressions, the activation condition is actually the negation of the masking effect which concerns any level of abstraction (requirements, code, or hardware) as explained by [13]. On example 2, the influence of In1 is masked if its activation condition (In2 and not In3) is not satisfied. Note that our definition, matches masking MC/DC as defined in [17] and [13]. This differs from [16]. Indeed, for X= Y and Z, the AC of the path Y to X is defined as Z in our definition: whereas it is defined as (not Y and Z) in [16].

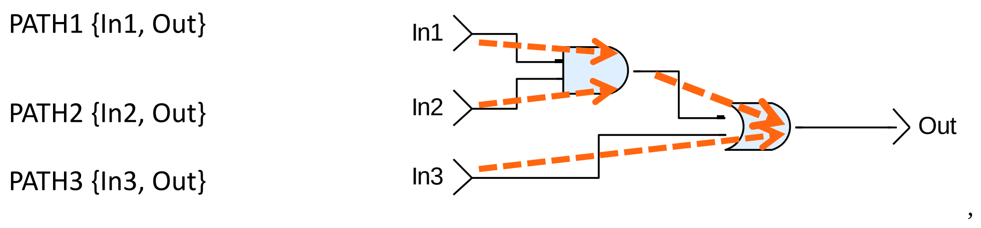


Figure 4: Example 2 Masking and Activation Condition in a Boolean Expression

The masking effect is also taken into account for coverage analysis of higher-level constructs such as state machines, where the influence of data depends on conditions related to the state, flows from/to states and transitions guards and priorities.

3.3 Position with Respect to Structural Coverage

Let us take example 1 to show the difference with structural coverage. The code that implements this model part may typically be implemented in three ways, shown in Table 2 in the form of pseudo-code.

Table 2: Implementation of Example 1

Implementation A	Implementation B	Implementation C

X = if_block(C2,B,D);
 	
Y = plus_block(X, E);
 	
Z = if_block(C1,A,Y);
	if (C2) {
	    X = B;
	  }
	  else {
	    X = D;
	  }
	  Y = X + E;
	
	if (C1) {
	    Z = A;
	  }
	  else {
	    Z = Y;
	  }
		if (C1) {
		    Z = A;
		  }
		  else {
		    if (C2) {
		      tmp = B;
		    }
		    else {
		      tmp = D;
		    }
		    Z = tmp + E;
		  }

Implementation A can be structurally covered just with just one execution cycle; it is obvious that such a test case does not cover the LLRs expressed in this model and is very poor. Implementation B can be covered just with the following set of test cases for C1/C2= {T/T, F/F}, which does not test the effect of B on Z.

The proposed model coverage criteria (even the most basic one) require at least C1/C2= {T/x, F/F, F/T} (where x stands for don’t care) in order to activate the effect of A, B, D and E on Z; more specifically, C1/C2= {F/F,F/T} shows the effect of B on Z . Note that this is also what is required for implementation C.

One can see that even on a very simple example, the proposed model coverage criteria capture the functional activation at model level regardless of the code that will implement it, whereas structural coverage, which is limited to control structures does not address functional flows aspects. This difference is of course amplified when analyzing more complex modeling constructs, such as complex data flow combinations or state machines.

In [18] it is shown that there is a subsumption (a kind of implication) hierarchy between data flow coverage and statement coverage. Intuitively, one understands that in order to activate a Definition-Use data flow, one needs to execute the related imperative statements in the generated C code, whereas execution of statements does not guarantee Definition-Use activation, as shown in the implementations A and B of Table 2. It has been verified in practice for complex models that tests covering the model also cover the code generated from that model, except few systematic cases which are predictable and justifiable. Formal demonstration of implication of code coverage by model coverage require some additional refinement of model coverage criteria (see conclusion of this paper).

3.4 Handling Structured Models

A Scade operator can be considered either as a black box or just as a syntactical/graphical device (it is said to be expanded or in-lined). This choice (which can expressed at model level as well as at code generation time) is a design choice. It is reflected by the code generator both in terms of causality analysis (i.e. any data shall be produced before being consumed) and in the architecture of the generated code: a non-expanded operator is implemented as a C function, whereas an expanded operator is in-lined as part of the function corresponding to the nearest non-expanded parent.

There are a number of possible approaches for handling such hierarchies such as:

a) Fully expand all operators of the model: this ideal analysis is supported by SCADE, and works for small models, but is impractical for large models both regarding the testing effort and machine resources.
b) Expand no operator at all, and consider each of them as pure black boxes. This was supported by historical versions of SCADE.
c) Selectively expand at model level consistently with the expansion selected for code generation and abstract non-expanded operators by considering their interfaces as atomic. This is supported by SCADE R16 and is the most typical use case.
d) The definition provided in [19] proposes a definition that abstracts non-expanded operators with a fine-grained dependency between their inputs/outputs.
e) Abstract specific operators with user-defined criteria. For instance for the integrator shown on Figure 5 , each instance of the integrator can be abstracted from its caller perspective as four objectives to cover: Reset, Normal integration, Low saturation and High Saturation. This analysis, supported by SCADE, is well suited to library operators.

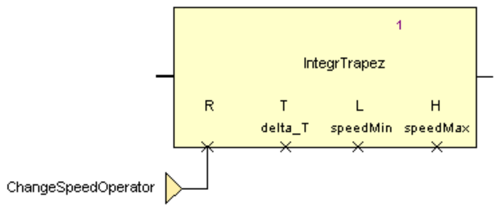


Figure 5: Example 4 Integrator Library Component

In the case of example 3, if node NASA_Example1 is not expanded, there are two Boolean expressions with 3 non-independent conditions each, if it is expanded there is one single expression with 5 conditions.

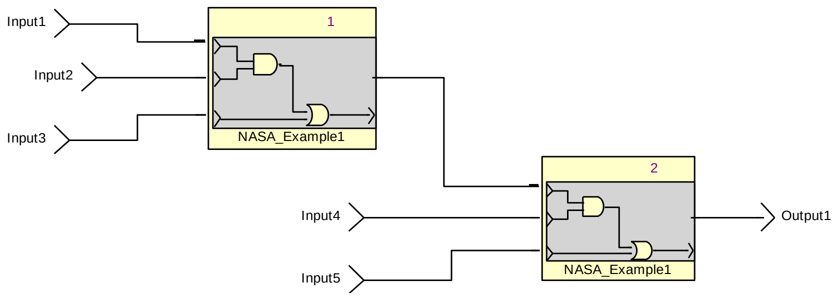
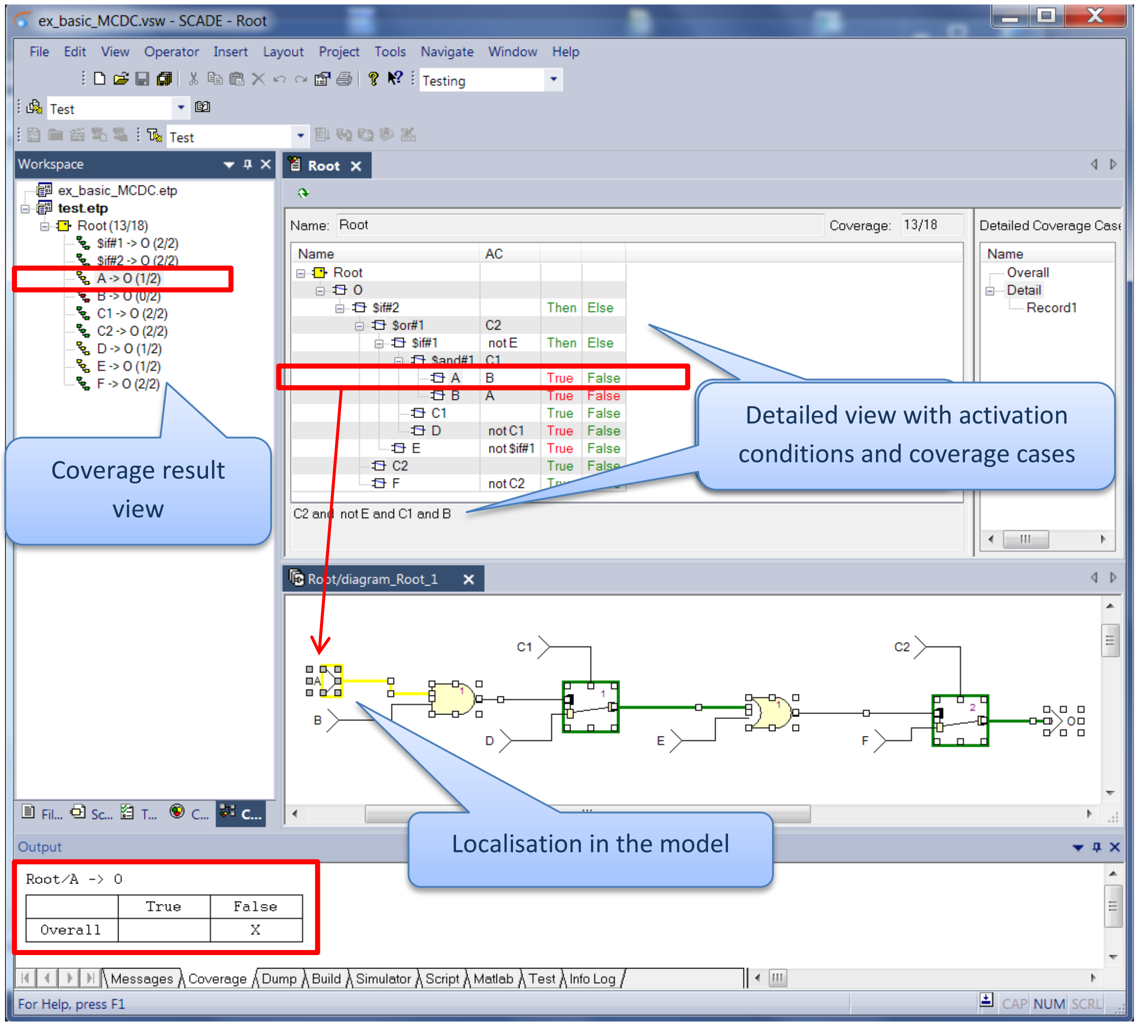


Figure 6: Example 3 Cascaded Operators with Boolean Expressions

4 Tool Support Overview

A toolchain called MTC (for Model Test Coverage) version 6.4.7 supporting of the above described criteria has been developed. Using appropriate observers, it runs test cases and collects model coverage information during simulation. The coverage results can be displayed in graphical form (e.g. Figure 7) or textual form. For applicants wanting to provide direct evidence of structural coverage, there is also a feature supporting direct measurement of structural code coverage of the source code that will be embedded, under execution of the same test cases as those used for model coverage analysis and with the corresponding coverage criteria (DC, MC/DC). The toolset has been qualified according to Tool Qualification Level TQL-5 of DO-330 [20].


ex_basic_MCDC.vsw - SCADE - Root
File Edit View Operator Insert Layout Project Tools Navigate Window Help
Testing
Test
Workspace
Root
Root/diagram_Root_1
Output
Coverage result view
Detailed view with activation conditions and coverage cases
Localisation in the model
Detailed Coverage Case
Overall
Detail
Record1
Messages Coverage Dump Build Simulator Script Matlab Test Info Log
For Help, press F1

ex_basic_MCDC.etp
test.etp
Root (13/18)
$if#1 -> O (2/2)
$if#2 -> O (2/2)
A -> O (1/2)
B -> O (0/2)
C1 -> O (2/2)
C2 -> O (2/2)
D -> O (1/2)
E -> O (1/2)
F -> O (2/2)

Name: Root Coverage: 13/18

Name	AC		
Root			
O			
$if#2		Then	Else
$or#1	C2		
$if#1	not E	Then	Else
$and#1	C1		
A	B	True	False
B	A	True	False
C1		True	False
D	not C1	True	False
E	not $if#1	True	False
C2		True	False
F	not C2	True	False

C2 and not E and C1 and B

```
Root/A -> O

         | True | False
Overall  |      |   X
```

Figure 7: Model Coverage Browser

5 Industrial Return of Experience

Model coverage analysis tools based on simpler criteria have been used since 2005 on many industry projects. The first versions supporting the new criteria described in this paper have been released by mid-2014 and used recently in several industrial projects in avionics and nuclear industry. The current version is MTC 6.4.7a.

5.1 MTU Experience Example

MTU develops emergency diesel engines for nuclear power plants. This function is very critical in case of failure of the electrical network. MTU develops the digital engine control unit for such engines. This unit shall in particular maintain safe state requested by user. The applicable standard are IEC60880 and IEC61508:2010 (SIL 3 for the software).

The previous experience of MTU about coverage concerns:

- Structural coverage with Cantata++
- Scade coverage with earlier versions of MTC:

The new MTC has been in use for more than a year (but not continuously). The main conclusion are the following:

- There is short learning time,
- It allows inspection of missing activation conditions and paths,

- There are potential improvement in the presentation of missing activations, since the reason of missing activations is sometimes hard to understand.

5.2 General Return of Experience

The new MTC has been used on complex airborne software systems (FADEC, flight control sub systems). The main feedback is the following:

- Model coverage analysis supports detection of insufficient testing and dead or deactivated model elements. The detection can be done early in the software lifecycle, which reduces cost and time for software development.
- There is a focus shift from code level to model level both for design and coverage analysis.
- There is a paradigm change from the traditional imperative mindset (statements, branching) to a data flow mindset (definition/use), which is more efficient but requires appropriate user training.
- The tools were able to handle complex software (3000 user operators, 100 000 model coverage points, 50 000 code coverage points). The ability to handle efficiently incremental software changes will be improved. The user interface will be improved for more comfortable navigation in large data sets.
- It is recommended to perform model coverage analysis early in the lifecycle, in order to detect and fix dead and/or non-optimal model elements and take full benefit of model coverage analysis.

6 Conclusion and Future Work

Model coverage analysis was a pioneering approach 10 years ago. It is now explicitly required and/or recognized as an alternate means for checking testing activities completeness in domains such as avionics (DO-331) or automotive (ISO 26262). The combination of the Scade language and of SCADE tools with appropriate model coverage criteria described in this paper allows software engineers to consistently shift their focus from code level to model level. Usage on large industry projects showed that it is an efficient means for detecting insufficient testing and dead or deactivated model elements.

Future work will provide additional benefits. For instance further refinements for addressing numeric aspects would allow performing analysis of singular points. Handling delays such as in [21] would allow assessment of sequential logic. Providing formal evidence of model to code coverage implication would allow application of DO-331 FAQ#11, definitely eliminating need to handle double-check of structural coverage.

7 References

[1] DO-178C, "Software Considerations in Airborne Systems and Equipment Certification DO-178C," RTCA Inc, 2011.

[2] CENELEC, EN 50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011.

[3] IEC, IEC_61508 Functional safety of E/E/PE safety-related systems, IEC, 2010.

[4] ISO, ISO 26262 Road vehicles – Functional safety, IEC, 2012.

[5] ECSS, "Space product assurance Software product assurance, ECSS-Q-ST-80C," 2009.

[6] IEC, IEC 60880 Nuclear power plants Instrumentation and control systems important to safety Software aspects for computer-based systems performing category A functions, IEC, 2006.

[7] DO-331, "Model-Based Development and Verification Supplement to DO-178C and DO-278A DO-331," RTCA Inc., 2011.

[8] DO-248C, "Final report for clarification of DO-178C Software Considerations in Airborne Systems and Equipment Certification DO-248C," RTCA Inc., 2011.

[9] Esterel Technologies, Scade 6.0 Language Reference Manual KCG-SRS-007, 2014.

[10] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic and R. De Simone, "The Synchronous Languages 12 Years Later," *Proceedings of the IEEE,* vol. 91, no. 1, 2003.

[11] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, "The synchronous dataflow programming language Lustre," *Proceedings of the IEEE,* vol. 79, no. 9, p. 1305–1320, September 1991.

[12] J.-L. Colaço, B. Pagano and M. Pouzet, "A Conservative Extension of Synchronous Data-flow with State Machines," in *ACM International Conference on Embedded Software (EMSOFT'05)*, 2005.

[13] J. J. Chilenski, "An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion," FAA Tech Center Report DOT/FAA/AR-01/18, 2001.

[14] CAST, "Position Paper CAST-10 What is a “Decision” in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)?," Certification Authorities Software Team, 2002.

[15] L. Clarke, A. Podgurski, D. J. Richardson and S. J. Zeil, "A Fomal Evaluation of Data Flow Path Selection Criteria," *IEEE Transactions On Software Engineering,* vol. Vol. 15, no. No. 11, November 1989.

[16] A. L. a. I. Parissis, "Structural test coverage criteria for Lustre programs," in *10th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, 2005.

[17] RTCA, "Software Considerations in Airborne Systems and Equipment Certification DO-178C," RTCA Inc, 2011.

[18] P. Frankl and E. Weyuker, "An Applicable Family of Data Flow Testing Criteria," *IEEE Transactions On Software Engineering,* vol. Vol. 14, no. 10, October 1988.

[19] V. Papailiopoulou, A. Rajan2 and I. Parissis, "Structural Test Coverage Criteria for Integration Testing of LUSTRE/SCADE Programs," in *FMICS'11 Proceedings of the 16th international conference on Formal methods for industrial critical systems*, 2011.

[20] DO-330, "Software Tool Qualification Considerations DO-330," RTCA Inc., 2011.

[21] M. Whalen, G. Gay, D. You, M. P. Heimdahl and M. Staats, "Observable Modified Condition/Decision Coverage," in *ICSE 2013*, San Francisco, CA, USA, 2013.

Applying Model-Based Techniques for Aerospace Projects in Accordance with DO-178C, DO-331, and DO-333

U. Eisemann, dSPACE GmbH, Paderborn, Germany

1. INTRODUCTION

The main difference between the new standard for software development in civil aviation, DO-178C (see [1]), and its predecessor, DO-178B, is that the new one has standard supplements that provide a greater scope for using new software development methods. The most important standard supplements are DO-331 (see [2]) on model-based development and model-based verification and DO-333 (see [3]) on the use of formal methods, such as model checking and abstract interpretation. These key software design techniques offer enormous potential for making software development in the aerospace sector highly efficient. At the same time, they not only maintain the high quality and observe the safety requirements for software, but actually improve them. These methods are seamlessly integrated in the development scheme of DO-178C, which is shown in a simplified form in Figure 1.

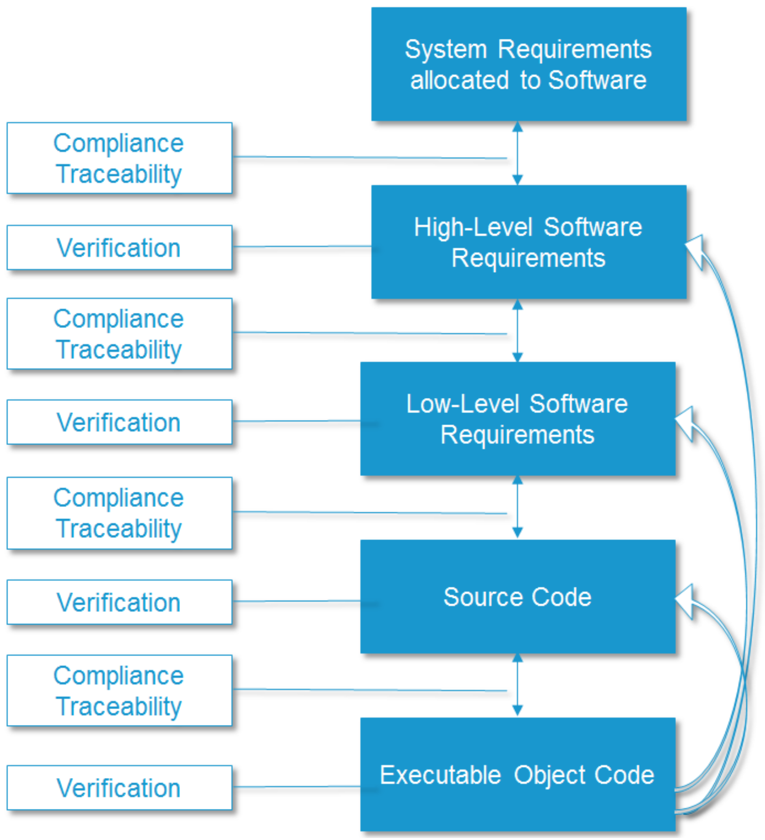


Figure 1: Important design and verification activities according to DO-178C (architecture design and verification have been omitted).

2. OVERVIEW OF THE MODEL-BASED TOOL CHAIN FOR DO-178C, DO-331, AND DO-333

This article describes how to use a model-based tool chain including Simulink®/Stateflow® ([4]), dSPACE TargetLink® ([5]), and tools by BTC Embedded Systems ([6]) to develop software right up to DO-178C Level A by using the standard supplements DO-331 and DO-333, see [7].

The main components of the tool chain are:

- Simulink/TargetLink for the graphical, model-based development environment
- BTC EmbeddedSpecifier (optional) for formalizing requirements
- TargetLink for automatic production code generation
- BTC EmbeddedTester and BTC EmbeddedValidator for meeting different verification and testing objectives

The above tools cover the following essential steps in the software development process in accordance with DO-178C/DO-331, see Figure 2:

- Specifying high-level requirements in the form of Simulink/TargetLink models, which then constitute specification models according to DO-331
- High-level requirements in the form of specification models can also be developed by using EmbeddedSpecifier, which is particularly attractive for converting existing textual requirements to formal requirements
- Representing low-level requirements in the form of Simulink/TargetLink models, which thereby constitute design models according to DO-331
- Automatically generating source code with TargetLink in order to convert the Simulink/TargetLink design models directly to high-quality ANSI C code
- Achieving various verification objectives of DO-331 via model-based testing with BTC EmbeddedTester and BTC EmbeddedValidator to efficiently create requirements-based test cases, automatically execute them in the Simulink/TargetLink environment, and determine coverage metrics such as MC/DC at the code level and model coverage to determine requirements coverage.
- Using BTC EmbeddedValidator for model checking as a formal method in the sense of DO-333 to demonstrate that the models comply with formalized requirements developed using BTC EmbeddedSpecifier.

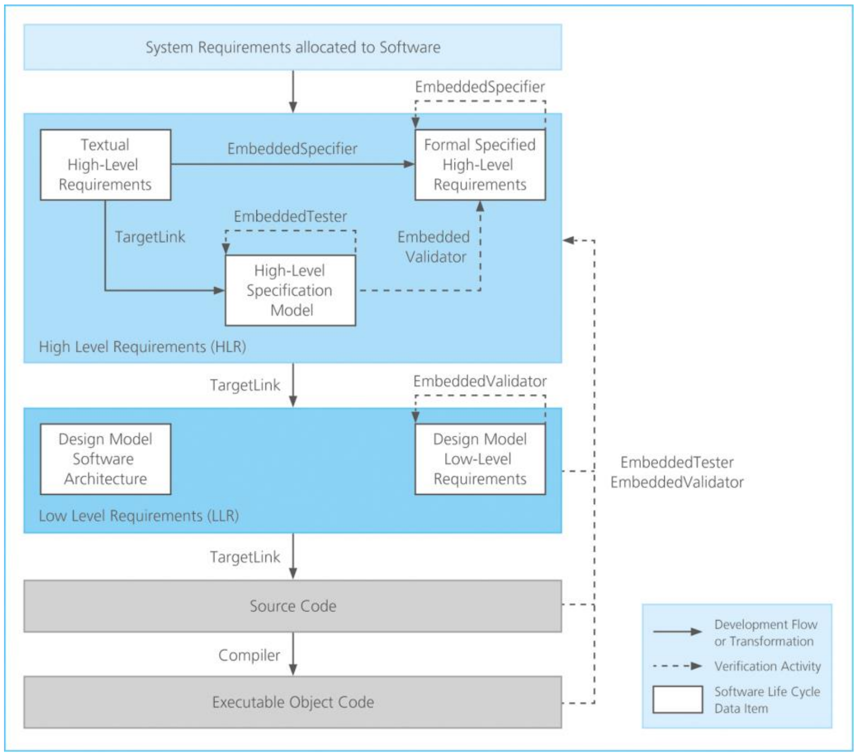

System Requirements allocated to Software
EmbeddedSpecifier
Textual High-Level Requirements
EmbeddedSpecifier
Formal Specified High-Level Requirements
TargetLink
EmbeddedTester
Embedded Validator
High-Level Specification Model
High Level Requirements (HLR)
TargetLink
EmbeddedValidator
Design Model Software Architecture
Design Model Low-Level Requirements
EmbeddedTester
EmbeddedValidator
Low Level Requirements (LLR)
TargetLink
Source Code
Compiler
Executable Object Code
Development Flow or Transformation
Verification Activity
Software Life Cycle Data Item

Figure 2: Model-based design tool chain for DO-178C/DO-331-compliant development.

3. MODELS AS A DOOR-OPENER FOR EFFICIENT SOFTWARE DEVELOPMENT

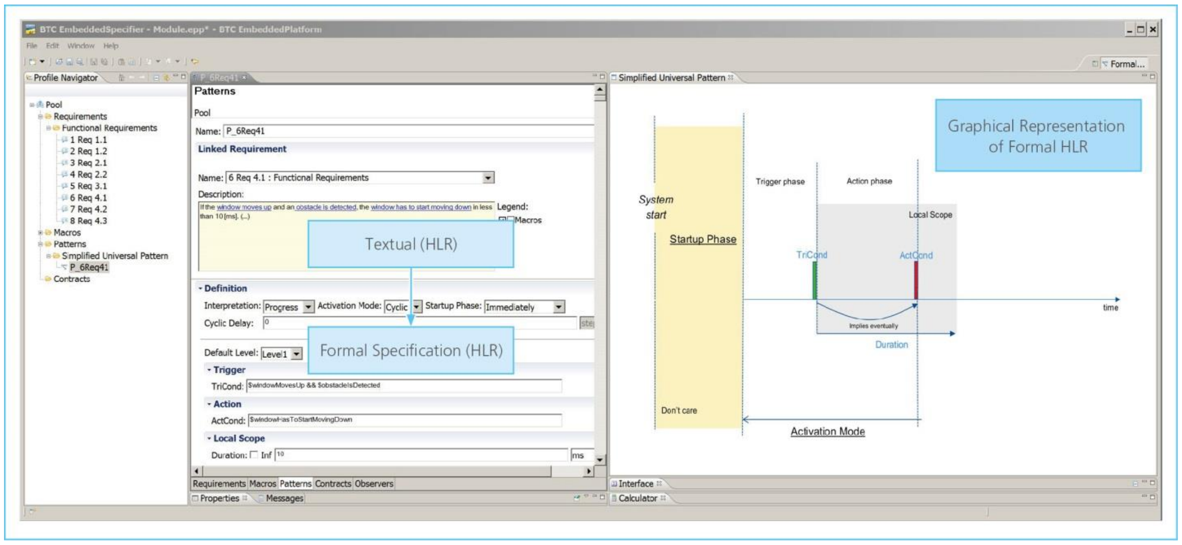
A key milestone for efficient and high-quality software development is representing requirements and specifications by models according to DO-331. A transition from purely textual to formalized requirements in the form of models opens up a wealth of possibilities for automated analysis, source code generation and verification, as will be demonstrated in this article.

Software requirements in DO-178C exist in two different forms, either as
• **High-level requirements (HLR)**
Simply put, they describe what the software is supposed to do but not how it should do it ("black box" view of the software). HLRs are derived from requirements for the entire system or subsystem, which are set up in the system process, e.g., as described in ARP 4754.
• **Low-level requirements (LLR)**
They describe the inner workings of the software, i.e., how the software is supposed to implement the HLRs ("white box" view of the software) and they must comply with the HLRs. It must be possible to translate LLRs directly into source code, which is later translated into the executable object code.

Models in the sense of DO-331 can now be used to represent requirements on these two levels (models for software architecture, e.g., in the form of UML models, are not discussed here but are widely used in practice and also covered by DO-331). In the case of HLRs, DO-331 speaks of specification models, whereas models for representing LLRs are referred to as design models.

3.1. Specification Models for High-Level Requirements

For some functional HLRs, it is best to use dedicated tools, such as BTC EmbeddedSpecifier, that support specific formalized patterns for expressing requirements. This is particularly the case when requirements are not written as a set of mathematical equations or “pseudo code” but come in the form of rather simple signal dependencies and conditions. BTC EmbeddedSpecifier is specifically geared towards requirements of such a form. Therefore, a very generic requirements design pattern in the form of a state machine is used which includes trigger conditions for the requirement as well as actions (see Figure 3). Timing dependencies can be expressed in the requirements design pattern as well. The tool is particularly helpful for the step-by-step transformation of informal textual requirements to formalized patterns in an intuitive process. For this purpose, informal signal names are replaced by actual interfaces in Simulink/TargetLink models and the meaning of the text is translated into the requirements pattern. The tool thereby simplifies the transition from informal to formal requirements with a clearly defined syntax and semantics, which lets users later use verification methods like model checking and automatic test case generation. Moreover, due to the formal nature of the requirements pattern, there is a precise definition for “requirements coverage” to identify whether developed test cases have covered all requirements.



Patterns

Pool

Name: P_6Req41

Linked Requirement

Name: 6 Req 4.1 : Functional Requirements

Description:
If the window moves up and an obstacle is detected, the window has to start moving down in less than 10 [ms]. (...)

Legend: Macros

Definition

Interpretation: Progress Activation Mode: Cyclic Startup Phase: Immediately
Cyclic Delay: 0

Default Level: Level1

Trigger
TriCond: $windowMovesUp && $obstacleIsDetected

Action
ActCond: $windowHasToStartMovingDown

Local Scope
Duration: ☐ Inf 10 ms

Figure 3: Converting informal textual requirements into formalized requirements.

HLRs can also be expressed by block diagrams and state diagrams in the form of Simulink/TargetLink models (see Figure 4). This is particularly useful if such models are already available from the system process. TargetLink automatically makes sure that only a “safe subset” of Simulink and Stateflow is used during the design process to avoid problems with certain modeling constructs. Moreover, compliance with company-specific standards for specification models can be verified by using automatic style checkers.

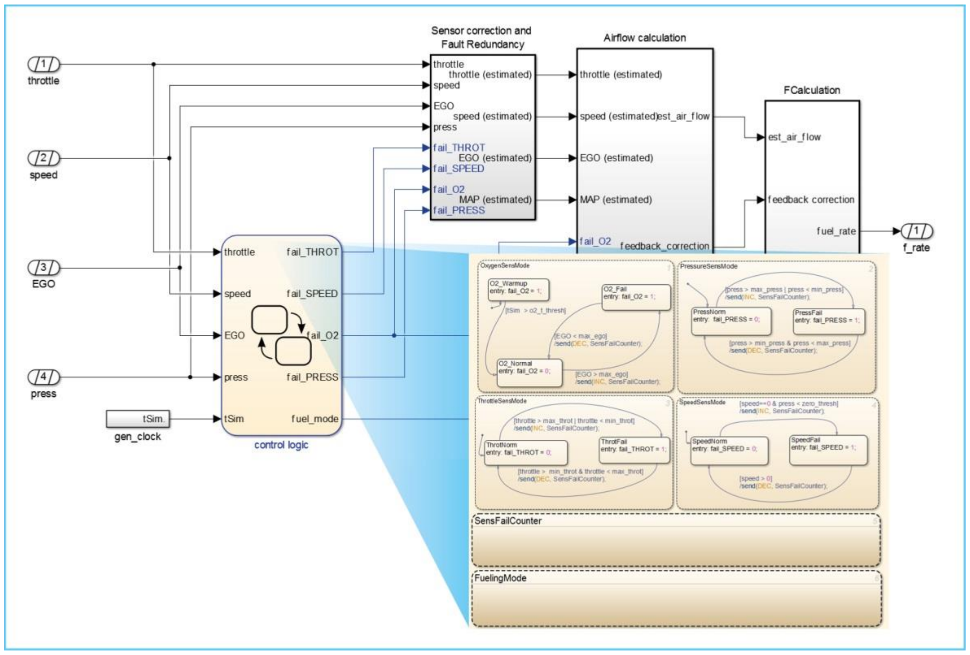


Figure 4: Simulink/TargetLink specification models for representing high-level software requirements.

3.2. Design Models for Low-Level Requirements

By nature, representing low-level requirements (LLRs) in the form of Simulink/TargetLink models is particularly popular, as source code can be generated from them automatically in subsequent steps (Figure 5). Such design models not only describe the actual functionality but also the necessary details of the software, such as internal data structures, distribution across different functions, control flow information, and, in some cases, fixed-point representations.

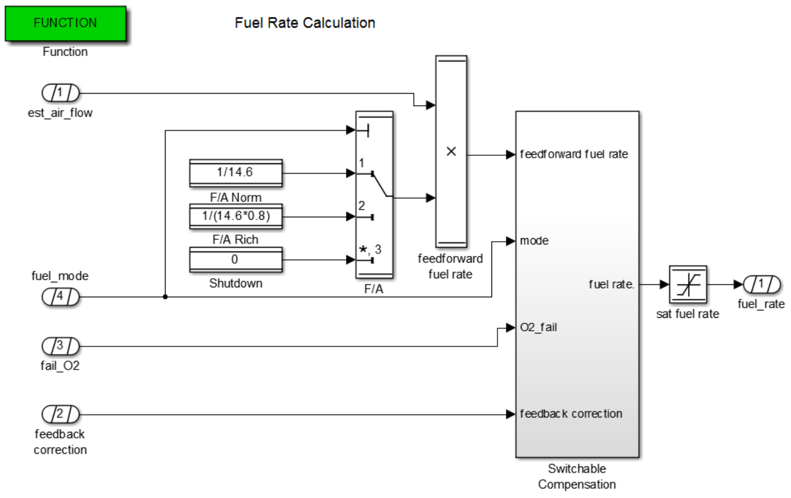


Figure 5: Design models in Simulink/TargetLink represent low-level requirements.

4. GENERATING SOURCE CODE FROM DESIGN MODELS

Design models representing low-level requirements offer a direct route to creating the source code – by using automatic code generation instead of manual coding. Automatic code generators, such as TargetLink, that transform Simulink/TargetLink models directly into ANSI C code are far superior in terms of quality and reliability compared to human programmers. The source code they produce

- Is generated deterministically
- Is very readable and suitable for review. This is ensured by extensive source code commenting, easily understandable symbol names and the use of a proper subset of the C programming language.
- Can be traced back directly to the individual parts of the design model from which the code was generated.
- Contains references to the requirements of the design model. This increases requirement traceability, which represents an important part of any software development process.
- Is highly configurable in terms of how to generate code, e.g., to meet company-specific coding guidelines and to integrate easily with the interfaces of the software architecture, library functions and legacy code.
- Is approximately as efficient as handwritten code. Individual optimizations on a detailed level let users specify the extent to which the code will be optimized, ranging from completely unoptimized to fully optimized code.

In addition to generating the actual source code, the code generator can generates other items, thereby assuring consistency between all the artifacts. Typical examples include:

- Additional documentation files adjusted specifically to special purposes
- Files with traceability information in order to support reviews and analysis of the source code
- Information on the requirements implemented by the source code to build a requirements traceability matrix

Using TargetLink for automatic production code generation from Simulink/Stateflow models has been widely used for many years, especially in the automotive sector, but also in DO-178B Level A projects (see [8]).

5. INNOVATIVE TECHNIQUES FOR VERIFICATION

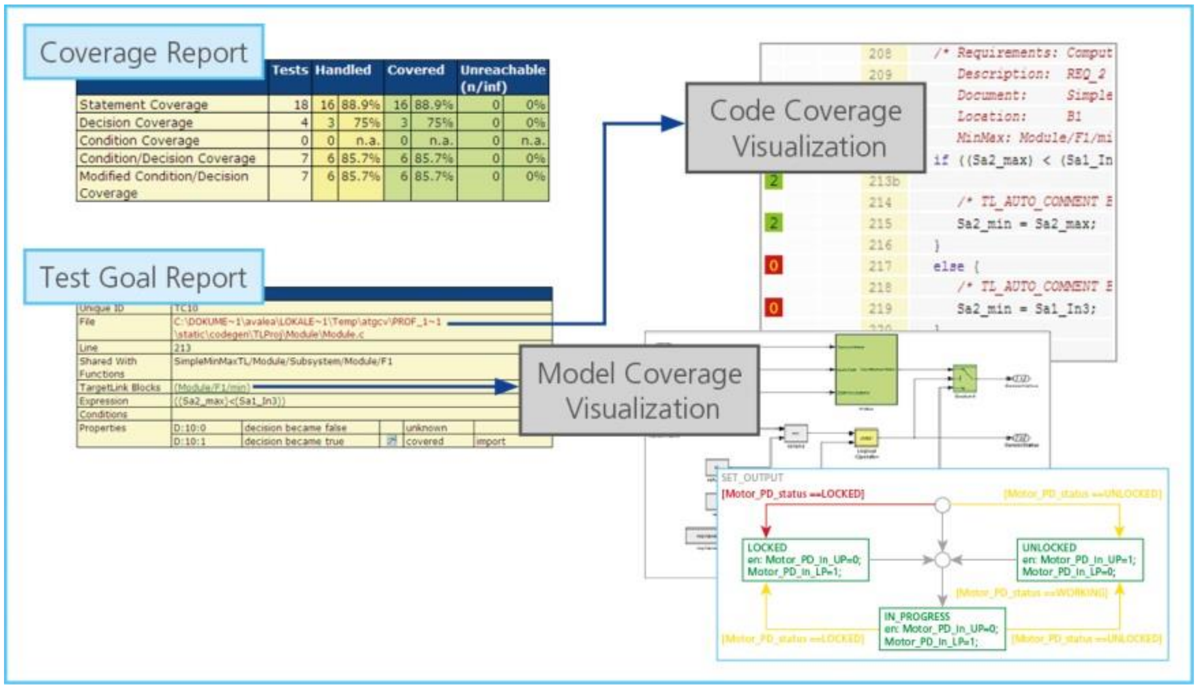
The great advantages of using models to specify requirements (HLRs and LLRs) are evident not only in automatic production code generation but also in other areas, such as verification. Particularly important activities are:

1. Verifying that models for HLRs and LLRs accurately implement the requirements from which they were developed
2. Verifying that models and source code comply with specific modeling guidelines and coding guidelines
3. Verifying that the source code exactly implements the requirements contained in the design model
4. Verifying that the executable object code implements HLRs and LLRs

The model-based approach combined with the formalization of the requirements in this case provides very powerful mechanisms to facilitate the verification steps, as will be shown in the following subsections.

5.1. Verifying Requirement-Model Compliance

A combination of model simulation, coverage analysis, and test case generation is especially well-suited for verifying the compliance between the requirements and the models that implement them. DO-178B and DO-178C state that test cases have to be created solely on the basis of requirements whose definitions directly include the required result. If a requirement is itself expressed as a model, e.g., a Simulink/TargetLink model, techniques for automatic test case generation, such as those provided by BTC EmbeddedTester, can be used to automatically generate test cases from the specification model (see [10]). If high-level requirements were expressed formally in BTC EmbeddedSpecifier, then automatic test case generation can also be used for those. Naturally, test cases can also be developed manually or semi-automatically based on the requirements. To check the compliance of a model against the requirements from which it was developed, the previously generated or developed test cases/stimulus values are executed with the model by means of model simulation. The Simulink/TargetLink/BTC environment makes it easy to run the model simulations and check the simulation results against the previously developed test cases. Moreover, model coverage analysis is used to investigate whether the various model elements are all covered completely (Figure 6) in order to identify undesired functionality in the model.


Coverage Report
Test Goal Report
Code Coverage Visualization
Model Coverage Visualization
SET_OUTPUT
[Motor_PD_status ==LOCKED]
[Motor_PD_status ==UNLOCKED]
LOCKED
en: Motor_PD_In_UP=0;
Motor_PD_In_LP=1;
UNLOCKED
en: Motor_PD_In_UP=1;
Motor_PD_In_LP=0;
[Motor_PD_status ==WORKING]
IN_PROGRESS
en: Motor_PD_In_UP=0;
Motor_PD_In_LP=1;
[Motor_PD_status ==LOCKED]
[Motor_PD_status ==UNLOCKED]

	Tests	Handled		Covered		Unreachable (n/inf)	
Statement Coverage	18	16	88.9%	16	88.9%	0	0%
Decision Coverage	4	3	75%	3	75%	0	0%
Condition Coverage	0	0	n.a.	0	n.a.	0	n.a.
Condition/Decision Coverage	7	6	85.7%	6	85.7%	0	0%
Modified Condition/Decision Coverage	7	6	85.7%	6	85.7%	0	0%

Unique ID	TC10				
File	C:\DOKUME~1\avalea\LOKALE~1\Temp\atgcv\PROF_1~1 \static\codegen\TLProj\Module\Module.c				
Line	213				
Shared With Functions	SimpleMinMaxTL/Module/Subsystem/Module/F1				
TargetLink Blocks	(Module/F1/min)				
Expression	((Sa2_max)<(Sa1_In3))				
Conditions					
Properties	D:10:0	decision became false		unknown	
	D:10:1	decision became true		covered	import

```
    208      /* Requirements: Comput
    209         Description:  REQ_2
                Document:     Simple
                Location:     B1
                MinMax: Module/F1/mi
             if ((Sa2_max) < (Sa1_In
2   213b
    214         /* TL_AUTO_COMMENT E
2   215         Sa2_min = Sa2_max;
    216      }
0   217      else {
    218         /* TL_AUTO_COMMENT E
0   219         Sa2_min = Sa1_In3;
```

Figure 6: Model coverage and code coverage reports.

Although the simulation capabilities of models help verify the compliance between requirements and models, the traditional execution of even a high number of test cases is always incomplete in a certain sense. If a proof is desired or all requirements are to be verified simultaneously during all simulation runs, different verification methods can be used as long as the requirements were specified formally in BTC EmbeddedSpecifier. By using the model checking functionality of BTC EmbeddedValidator, users can prove or disprove compliance between the formally specified requirements and the design model. In particular, the model checking engine of EmbeddedValidator is fully capable of handling not just boolean and integer data types but also floating-point design models. Model checking is one of the techniques considered in the formal methods supplement DO-333 of DO-178C. Providing a proof of correctness is of course very appealing, but the usual limitations of model checking need to be considered, like a potential explosion of the state space with increasing complexity of the design model. If a model checking approach is not feasible, formal requirements specifications can nonetheless be very helpful to generate requirement observers, which verify all requirements simultaneously during all simulation runs.

5.2. Compliance of Models and Code with Modeling and Coding Standards

In order to check whether models comply with their respective modeling standards, style checkers such as MXAM from Model Engineering Solutions (see [11]) or the Simulink Model Advisor are typically used. These tools support automatic guideline checking in order to simplify review activities on the different models. The compliance of the generated source code with coding guidelines is usually verified by using static source code analysis tools that are available on the market.

5.3. Compliance of Source Code with Design Models

In order to check that the automatically generated source correctly implements the design models from which code was generated, developers can use reviews and analyses, which is also facilitated by TargetLink. In order to simplify code reviews, traceability information as well as hyperlinks between design model elements and the respective generated source code lines are provided. In addition, further analyses can be fully or partially automated.

5.4. Compliance of Executable Object Code with High-Level and Low-Level Requirements

A typical way to verify that the executable object code implements HLRs and LLRs according to Figure 1 is to execute it on the target platform. TargetLink provides powerful mechanisms for this in the form of processor-in-the-loop simulation, in which the automatically generated code is translated directly by the target compiler and executed on an evaluation board with the target processor (Figure 7). Testing is performed in the Simulink/TargetLink environment and can be directly compared with the results of a model simulation (model-in-the-loop simulation). This test design lets users reuse all test cases that were developed manually or generated automatically. BTC EmbeddedTester provides a powerful environment for performing the required tests, including the automatic comparison of model simulation and processor-in-the-loop simulation. The associated test reports are created completely automatically. BTC EmbeddedTester also supports code coverage analysis (MC/DC coverage, condition coverage, etc.), see Figure 6.

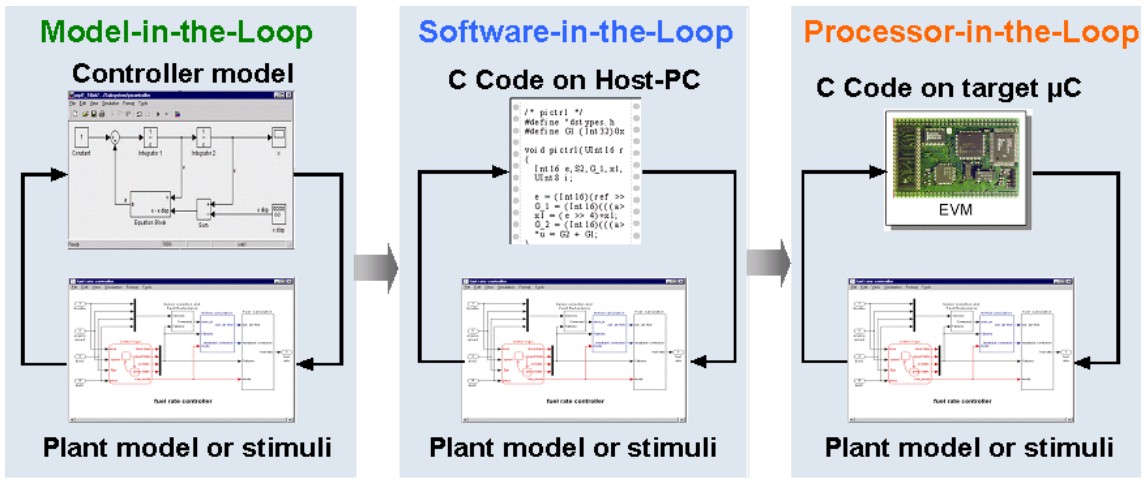


Figure 7: Model simulation and tests in different simulation modes.

6. TOOL QUALIFICATION ASPECTS, LIMITATIONS AND EFFECTIVENESS OF THE TOOL CHAIN

The proposed tool chain is based on the Simulink/Stateflow modeling environment and TargetLink for automatic source code generation. But a code generator as powerful as TargetLink is very hard to qualify as a criteria 1 tool according to DO-330 (see [9]), the Software Tool Qualification Considerations supplement to DO-178C. Therefore, the idea is to use an unqualified code generator and instead perform verification steps as required by DO-178C/DO-331. However, the verification steps greatly benefit especially from the use of verification tools provided by BTC, be it for the source code or the object code. This means that tools like EmbeddedTester (criteria 3 tool) and the model checking engine in EmbeddedValidator (criteria 2 tool) would have to be qualified to claim certification credits for the application of the tools.

Regarding the application of the tool chain, the following primary aspects and limitations should be considered:

- Since the whole tool chain makes extensive use of Simulink/Stateflow, its application is primarily attractive where this environment seems most appropriate judged by the nature of the algorithms and the

software that is to be developed. Where algorithms are best described by block diagrams and state diagrams, the tool chain can boost efficiency.
- Another limitation lies of course in the optional application of model checking to verify the compliance between models and the requirements from which the models were developed. The larger the model and the more complicated the requirement, the less likely it is that a model checking engine will be able to provide a proof in time.

It should be noted that production code generators do not constitute a weakness in the tool chain. On the contrary, a modern production code generator like TargetLink produces code much more reliably and with much higher quality than a human programmer could.
In summary, the tools provide powerful mechanisms not only for requirements definition and design, but especially for the various verification steps, making it possible to meet the objectives of DO-178C quite easily and with less effort than by using conventional techniques.

References

[1] RTCA DO-178C “Software Considerations in Airborne Systems and Equipment Certification”. RTCA, Inc., 2011
[2] RTCA DO-331 “Model-Based Development and Verification Supplement to DO-178C and DO-278A”. RTCA, Inc., 2011
[3] RTCA DO-333 “Formal Methods Supplement to DO-178C and DO-278A”. RTCA, Inc., 2011
[4] MATLAB/Simulink/Stateflow, www.mathworks.com
[5] dSPACE TargetLink, www.dspace.com/go/targetlink
[6] BTC EmbeddedSpecifier, BTC EmbeddedTester, BTC EmbeddedValidator, www.btc-es.de
[7] “TargetLink – Model-Based Development and Verification of Airborne Software“, dSPACE GmbH Paderborn, July 2014
[8] Aloui, Andreas, “C Code Reaches New Heights at Nord-Micro”, dSPACE Newsletter, 1/2002
[9] RTCA DO-330 “Software Tool Qualification Considerations” Supplement to DO-178C and DO-278A”. RTCA, Inc., 2011
[10] H.J. Holberg, U. Brockmeyer, “Significant Quality and Performance Gains through Fully Automated Back-to-Back Testing”, EmbeddedWorld 2010
[11] “Model Examiner”, www.model-engineers.com

Introducing SCADE Model–Based Development into a Safety-Critical System Environment

Abstract

With the publishing of ED-12C and ED-218, an opportunity has been created in which Model-Based Development is better defined for a Safety Critical System Environment. This positioning paper describes the approach and methodology applied to move from a conventional development, to a Model-Based Development. The major issue of how to integrate the two development methodologies is discussed.

Introduction

Since the publication of ED-12B, advances and experience have been gained in model-based development and verification, their application, and supporting tools. As the use of this technology for critical software applications in avionics has increased, there are a number of issues that need to be considered to ensure the safety and integrity goals are met. With the ED-12C [1] edition and introduction of ED-218 [2] these issues are addressed.

For engine control systems the demand placed on the provider is that the software is produced to ED-12C Safety Critical Level A standard [1]. Section 1 of this paper considers the selection of a model-based development tool with a certified code generator which meets the requirements of ED-12C and ED-218. This removes the obligation to perform code reviews and low level testing in the V-model process [2]. Section 2 discusses the integration of the candidate Model-Based Development Tool (MBDT) with the existing technology that had been used over a series of control system developments. Section 3 addresses the tooling that currently supports the existing process. This poses the question of whether to update and integrate, or to generate specific tooling to support the MBDT. Finally Section 4 is concerned with other aspects of the introduction of a Model-Based Development into projects.

Section 1 – Selection and Process for the Model-Based Development Tool

The primary criterion for selecting a Model-Based Development Tool is that there is an ED-12C qualifiable code generator that produces ADA code from a model. A mandated process under ED-12B/C is to demonstrate, using independent review and test, that the code meets the low level design. A significant amount of resource is spent doing this, however very few coding errors are made, typically 2% against 70% in requirements. Under ED-12C if the code has been produced by a qualified code generator then the demonstration is no longer required. Investigating the availability of qualifiable code generators discovered that ANSYS/Esterel Technologies [3] markets such a product – SCADE Suite, as a member of a four part product set. SCADE Suite uses either KCG C qualifiable code generator, or a currently unqualifiable KCG ADA code generator (KCG ADA is undergoing a process which is expected to complete in 2015 which will result in a qualifiable version). Further investigation showed that there are no qualifiable ADA code generators available. SCADE Suite comes with a large set of utilities; however these are not qualifiable for reasons other than technology. SCADE Suite has a significant user base and, since Esterel Technology became part of ANSYS, the tool is expected to be available to support development over the life time of an engine.

A gated process to introduce a SCADE design process into the company was enacted. This not only examined all aspects of using SCADE for designing and modelling engine behaviour, but the business case as well. Part of the process is to have prototype and pilot projects to demonstrate that the SCADE methodology and tooling is robust and complete. An important part of the process is training, so a SCADE Tutorial was written for use in-house to be led by an internal tutor. The SCADE Tutorial emphasises the particular aspect of designs that are commonly used on projects. The training covered both design and model test coverage features of SCADE and is used in conjunction with the SCADE Suite examples and documentation.

Three 'Golden Examples' or exemplars, were developed by a trained implementation team using a previously implemented set of High Level Requirements (HLRs). They modelled a scheduler, a signal validation scheme and an engine status state machine. They were analysed against the original developments to validate the artefacts produced and compare metrics. The SCADE process identified a problem in a requirement specification which was corrected immediately. In the original implementation the problem had gone through a full development cycle before being identified and corrected. The full SCADE package was then exercised in a pilot project. A team of four engineers from a sister company were trained using our internally developed training material 'The SCADE Tutorial', and then given the same 'Golden Example' requirements to design and verify. This pilot team also discovered the specification problem and completed the task in a slightly longer time than the original team. Full ED-12C Level A processes were used in both cases and, because SCADE KGC ADA is not currently qualified, the code artefacts are subject to review. SCADE is now In-Service within the company and has been successfully used on three engine types for shaft break detection, air flow and lean burn components.

Section 2 – Integration of the Model-Based Development Tool and the Existing Process

The existing process uses Artisan Studio [6] for UML Modelling and development and we will always want an Artisan model to place the SCADE components into. This means that all SCADE components have to be integrated into our Fixed Priority Scheduler (FPS) for the Electronic Engine Controller (EEC). The software architecture is modelled in UML and each schedulable component in the model can only have "init" and "run" methods in each software component class. So a way had to be found to access SCADE through a single entry point so as to be able to integrate the SCADE component into the architectural model. A SCADE component stereotype allows the architecture modelling tooling to generate a SCADE/UML interface. A call is generated of the type "componentSCADE_execute" within the "component.run" method, passing a ".ctx" SCADE context object as a parameter. Also the data for the accessor methods to the boundary items between SCADE and the architecture model have to be passed as parameters, these are picked up as sensors within the SCADE package. Any test points are set up as probes and passed back through the ".ctx" SCADE context object. Having a single entry point means that requirement testing has to be behavioural testing in black box mode. Subsequently we discovered, for verification tooling and testing reasons, that multiple entry points might be preferable, if so then a "componentSCADE_execute" operation is needed for each one. However this should be carefully considered, due to the extra work involved.

Section 3 – Tooling for Two Methodologies

In the existing development environment, the processing of hand written ADA is handled by the tool and its code generator. This builds the component source file from all the architectural diagrams and the hand written code. The ADA is checked for SPARK compliance and complied by an in-house complier based on GNAT. At this point the component source files go through an ED-12C mandated review process to check that code meets design. Often the code is subjected to find and fix testing on a rig to expose and capture requirement, design and code problems. On completion of the review, the components are put into a build and undergo a test process. SCADE Suite provides a simulation and model test coverage MTC facility, thus the designer can model and simulate the design. This provides an early opportunity to discover if any problems exist in the component HLRs and, at the end of the design process, there is a high confidence in the validity of the model. After a design review the model code is transferred to the test process for testing against the HLRs.

In order to integrate SCADE Suite with the architecture model, a set of tools were developed. These are based on the work carried out in the 'Golden Examples' and the pilot activities. Previously altering the FPS was not an option, so a convention to name SCADE operations as "componentSCADE_execute" was made. The "component.run" method is designed exactly as before, but now with "componentSCADE_execute" method embedded, which passes, as a minimum, a ".ctx" SCADE context object as a parameter. The SCADE Integrator tool was developed to generate the SCADE component model according to the structure defined in the architecture model. This picks up all the Methods, Interfaces, Development Variables and Test Points in the component class and creates the SCADE Component Model. Any SPARK annotations are also generated. Data Tables are processed by a Graphical Data Loader tool which puts the data into graphical data packages in the architecture model for use by the software components.

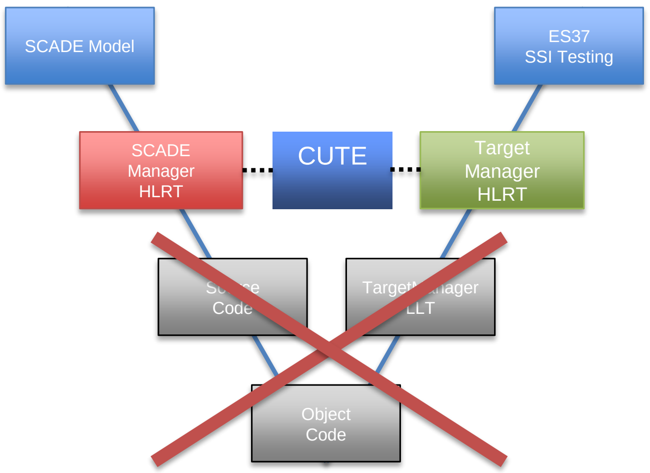
SCADE Suite operates in its own environment and holds all the type definitions in single file called "KCG_types". This is not an issue when there is a single "componentSCADE_execute" in the "component.run" method. However when there are more than one, then there is a problem with duplicated "KCG_types" files.

Another feature is that SCADE Suite creates its own variable names. For example a record structure "{delta, flt}" is named "daft_1". Understanding this algorithm is easy, however when it comes to Graphical Data with multidimensional arrays, naming proved to be impenetrable. The SCADE Code Processor tool was developed to structure/call KCG using command line arguments and to edit the graphic data files to integrate with KCG generated code files. For example each of the "KCG_types" is forced to generate as "componentSCADE_KCG_types", resolving the name space issue.

The testing tool used for the existing systems is ADATest. Our in-house Component Under Test Explorer tool (CUTE) is used to generate the necessary test cases to demonstrate coverage and robustness. The output ADATest is an ".ath" file which is executed to test the component and gather coverage from an instrumented version. CUTE was modified to access the architecture model "componentSCADE_execute" method and produce "componentSCADE.sth" files for processing by SCADE. There was no change to how CUTE was used. The SCADE Manager was developed to process the CUTE "componentSCADE.sth" files and generate ".sss" files and run SCADE MTC in command line mode. This generates coverage reports for the SCADE components.

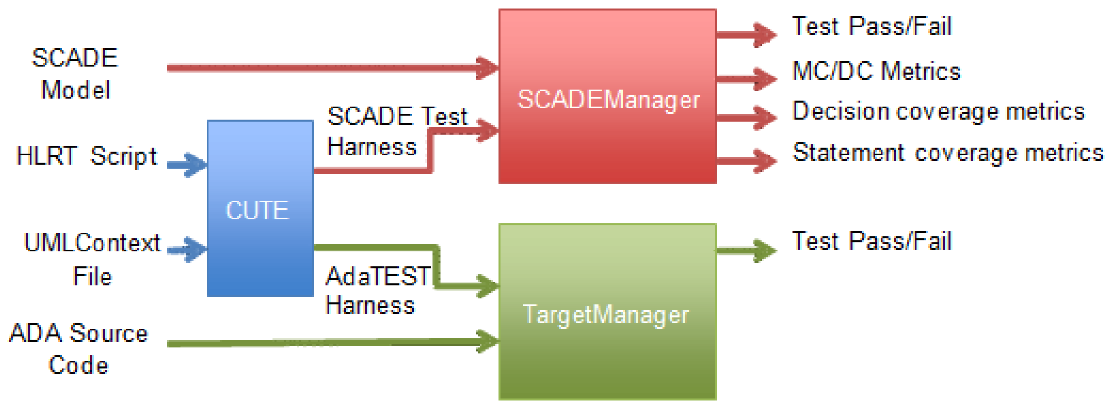
Model-Based Systems Engineering (MBSE) is used to produce HLRs. In order to provide traceability the SCADE LifeCycle Requirements Management tool is used. This uses regular expressions to process a range of documents, including ".pdf", ".doc" and ".xls", and also has an interface to DOORS. The requirement tags are extracted from the HLRs and allocated to the SCADE operations. A rich set of views and utilities are available in the tool to make requirement tracing easy. Finally the software design document generator tool was modified to integrate the SCADE design documentation produced by SCADE Suite.

The truncated SCADE V lifecycle under ED-12C and ED-218 is given in Figure 1, the SCADE Context diagram is given in Figure 2 and the conversion of the UML Component into SCADE is given in Figure 3.



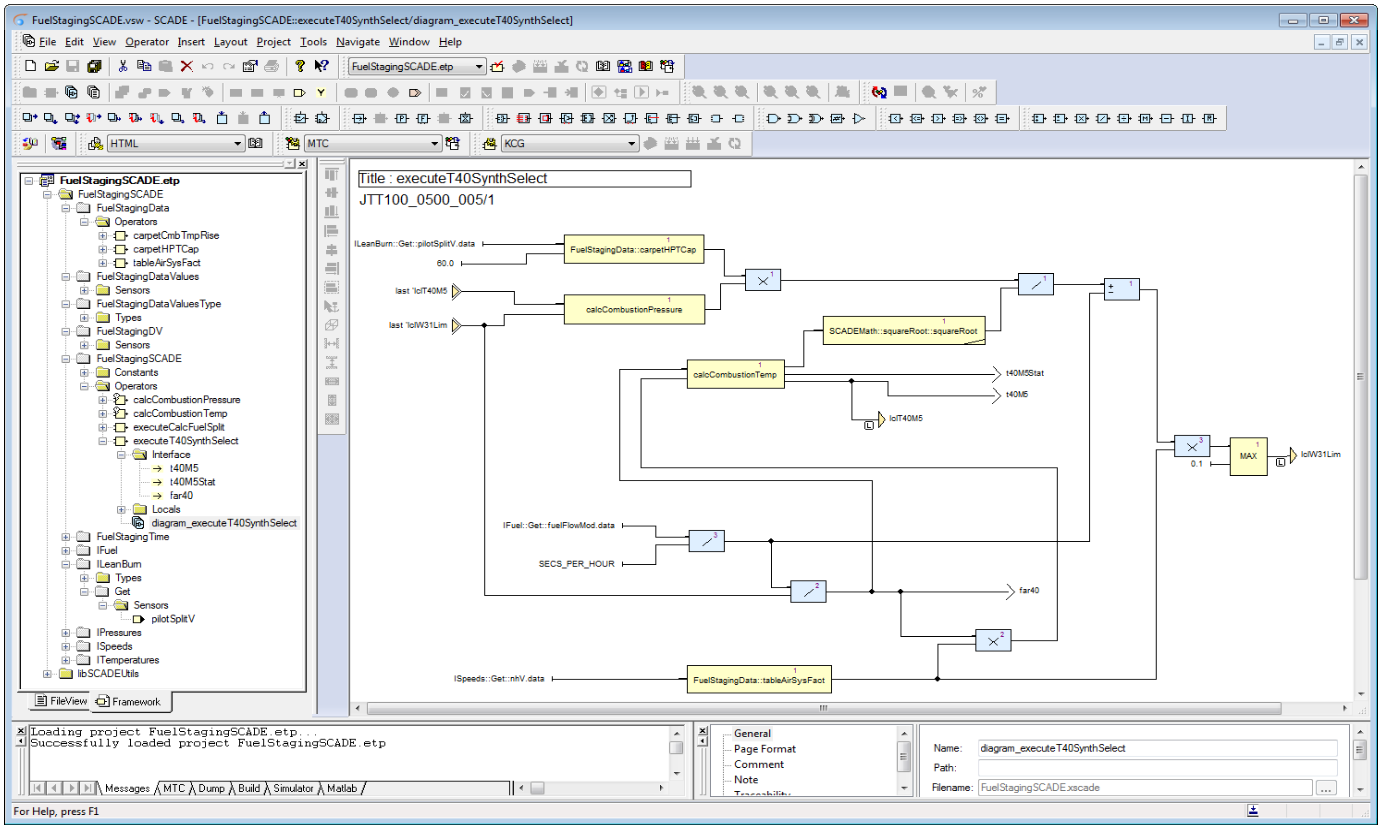
- SCADE generates qualified code which eliminates the need for Low Level Testing
- SCADE Verification Tooling consists of CUTE and SCADE Manager
- CUTE generates test harness files for SCADE (.sth) and Target (.ath)

Figure 1 - SCADE V Life Cycle



- SCADEManager provides the same functionality as Target Manager, but against the SCADE Model
- CUTE has been updated to generate SCADE Test Harness files (.sth)
- High Level Requirement Test (HLRT) Scripts are run against the SCADE Model and then transferred to the Tager System.
- SCADEManager executes the SCADE Test Harnesses.
- Target Manager executes the ADA TEST Harnesses.

Figure 2 - SCADE Context Diagram



```
Loading project FuelStagingSCADE.etp...
Successfully loaded project FuelStagingSCADE.etp
```

Name: diagram_executeT40SynthSelect
Path:
Filename: FuelStagingSCADE.xscade

Figure 3 - UML Component Structure Converted into SCADE by SCADE Integrator

<u>Section 4 – SCADE on Projects</u>

The major issue was the previous decision to use the KCG ADA code generator. This is currently being "made qualifiable" and will finally eliminate the need for code reviews. Until this happens they still have to be performed to comply with ED-12B/C for certification. Against using the qualified C generator was the necessity of providing ADA/C conversion wrappers for every SCADE call. In an environment where safety is critical and Worst Case Execution Time (WCET) has to be as low as possible, the wrappers were considered to be unnecessary overhead. Another factor was the interfaces between the layers of the architecture models. As SCADE is being used initially on a refresh program, the effort of re-designing the existing systems would have been too great for the benefit obtained. So introducing SCADE and integrating with existing software has to be as robust as possible and this burden falls on the tooling effort.

Critical to the success of the project was the development of SCADE and SCADE MTC Tutorials. These are complementary to those provided by SCADE Suite. The SCADE Tutorials are concerned with generating a set of SCADE projects from nothing and exploring each set of SCADE primitives in depth. This is especially true with maps and folds, which was found to be a difficult concept to grasp. Also the examples are tailored to meet typical types of project component.

By generating the 'Golden Examples', from a previously implemented HLR set, comparisons were able to be made between the two SCADE teams which found that there were no significant differences between the quality of the teams work or the time spent. This showed that the SCADE Tutorial and training is fit for purpose. The detection of the problem in the original HLRs helped demonstration of the value proposition of introducing the SCADE process.

A set of SCADE Utilities comparable with the existing Utilities were created. By looking across three existing projects only those which were in all three were generated. An exercise was carried to calculate the WCETs and they were compared with the existing times. The auto generated code was within 3% of the hand written code and in the binary search turned out to be faster. SCADE Suite is architected such that SCADE Utilities and project specific versions of Utilities can be deployed simultaneously. The intention is to make the SCADE Utilities product line software components once KCG ADA is qualified.

Due to the IT policy and against advice SCADE Suite was installed on a Citrix network. This proved impossible to run, and so it was installed on the local machine. This again proved to be a problem, due to SCADE Suite requiring to write logging files in its' program directory, which is against our IT policy. The solution turned out to be to run SCADE from a shortcut with the Start In parameter set to a writeable directory.

The look and feel of SCADE Suite was good, and the help system was exceptional. If anything it was over thorough, which is not often stated. The whole SCADE process took about 4 man years to achieve, and was rigorously progressed through a gated improvements process.

By using SCADE the dynamic of planning and project management is changed. The software spend is loaded upfront into the system and software interface and on into the design phase. Of course the cost of code review and low level testing is removed, but is taken up with design activity costs. As part of the ED-12C/ED-218 process there is also a need to re-execute the tests in target to show compatibility with target hardware with attached costs. When comparing SCADE development components against existing development a cost saving of about 18% was seen due to the removal of scrap and rework (S&R), where the components have been right first time. We made the benefit of S&R almost a break even cost because we knew that there were further benefits to be gained, but hard to quantify. As expertise increases other aspects of project spend will be reduced.

Another feature of using SCADE is the effect that it has on the levelling of requirements. There is a tendency to write requirements in a language that resembles pseudo code, and effectively imposes unnecessary constraints on the designers. Using SCADE diagrams to express design helps to encourage a more optimum level of high level requirements. A much greater interaction and discussion between the systems and software engineers was noticed as an evolved behaviour. From our initial experiences it is expected that further changes will occur in the way that projects are managed and resourced, which is a good thing in an environment of continuous improvement.

Conclusion

Ideally anyone wanting to use SCADE Suite should start on a new project that is not dependent on an existing system. ANSYS also provide a SCADE System product which integrates with SCADE Suite, which could have eliminated our architecture tool integration issues. However this should be compared against other vendors or indeed the merits of integrating with existing incumbent systems and processes. Since the code does not need to be inspected the selection of a specific language KCG based on source language preferences should not be an issue but where the code is to be integrated with other code the overheads of mixing languages must be considered. If SCADE Suite is to be used with existing systems then be prepared for lots of challenges. However with a robust plan, analysis, diligence and perseverance it can be achieved.

Acknowledgements

All this would not have been possible without the expert help of Edward Anguish, Steven Blyth (Ansys), Andy Burrell, Simon Daniel, Ian Hopkins, Dr Stuart Hutchesson, Yinan Jiang, David Merrifield, Andy Rayner and Tim Roughan.

Bibliography

[1]ED-12C, SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION

[2] ED-218, MODEL-BASED DEVELOPMENT AND VERIFICATION SUPPLEMENT TO ED-12C AND ED-109A

[3] SCADE Suite – Esterel Technologies - A Division of Ansys Corp

[4] SPARK – Altran-Praxis

[5] ADATest – QA Systems GmbH

[6] Artisan Studio – Atego a wholly owned subsidiary of PTC Inc.

Acronyms

SCADE – Safety Critical Application Development Environment

CUTE – Component Under Test Explorer

SPADE – Southampton Program Analysis Development Environment

SPARK – SPADE ADA Kernel

Authors

Dr Phil Birkin CEng MIET can be contacted at philip.birkin@controlsdata.com

Duncan Brown CEng FBCS Roll-Royce Fellow can be contacted at duncan.brown@controlsdata.com

Session 2

Multicore

Wednesday 27th, 11:00 – Guillaumet

Hard Real Time and Mixed Time Criticality on Off-The-Shelf Embedded Multi-Cores

Albert Cohen[*], Valentin Perrelle[†], Dumitru Potop-Butucaru[*], Marc Pouzet[*], Elie Soubiran[‡], Zhen Zhang[*]
[*]INRIA and DI, ENS, Paris, France,
[†]CEA, LIST, 91191, Gif-sur-Yvette, France,
[‡]Alstom Transport and IRT SystemX, France

***Abstract*—The paper describes a pragmatic solution to the parallel execution of hard real-time tasks on off-the-shelf embedded multiprocessors. We propose a simple timing isolation protocol allowing computational tasks to communicate with hard real-time ones. Excellent parallel resource utilization can be achieved while preserving timing compositionality. An extension to a synchronous language enables the correct-by-construction compilation to efficient parallel code. We do not explicitly address certification issues at this stage, yet our approach is designed to enable full system certification at the highest safety standards, such as SIL 4 in IEC 61508 or DAL A in DO-178B.**

***Index Terms*—Mixed criticalities, Multi-core, Embedded real-time system, Synchronous Language, Time-triggered execution**

I. Introduction

This paper presents the design of a synchronous language enabling hard real-time applications to run on off-the-shelf multi-core platforms.[1] The language and methodology ensure the isolation of time-critical tasks from the non-time-critical ones under the following three hypotheses:

1) most of the computational load takes place in non-time-critical tasks;
2) it is possible to program the reaction to the absence of timely data, when non-time-critical tasks are delayed;
3) the target multiprocessor provides means to strictly prioritize memory accesses of one or more processors executing time-critical tasks, or to fully isolate such accesses into a scratch-pad memory; and the target also supports asymmetric multiprocessing (e.g., bare-metal execution on one core and Linux on another).

We illustrate our approach and validate it on a train signaling use case provided by Alstom Transport. It is representative of the complexity in terms of vital/non-vital code interweaving, operational performance and availability constraints. The system function is called “Passenger Exchange” (PE). This function takes control of the train when safely docked at a station; it organizes the exchange of passengers (train and station doors opening/closing) while protecting them from any untimely train movement or non-aligned doors opening, and finally gives the departure authorization when all safety conditions are met. The functional specification is made of more than 300 requirements (natural language and SysML), and the system function is composed of about twenty sub-functions.

The PE application is partitioned into tasks, some of them being safety-critical and hard real-time, and some of them being mission-critical but non-vital. These tasks expose input and output signals and may result from the compilation of a synchronous block-diagram language. Unlike most related work on mixed criticality [2], *dependences and communication among tasks of different criticality are allowed*. There is a simple reason why we can afford such a breach of criticality partitions: our approach composes tasks of different *time*-criticalities, *without relaxing any other validation requirement*. In other words, *all tasks may still be certified at the highest (relevant) level of safety*, but we acknowledge that *only a subset of the tasks needs to be time-predictable and validated against real-time constraints*. Let us discuss this hypothesis on the PE application. The computation of the doors that are safe to open (e.g., because they are not aligned) is safety-critical, as well as the task preventing train departure if safety conditions are not met (e.g. the doors are open or opening). Less vital tasks are in charge of operating doors with respect to the mission and to a time table; these are still mission-critical since the quality of service depends on the rare occurrence of timeouts. In this example we identify two occurrences of mission-critical to safety-critical communication. First, in order to ensure that door commands (mission-critical) do not lead to an accident, they must be checked against the enabled set of doors (safety-critical). Second, the departure authorization (safety-critical) must be computed regarding door commands to ensure that no opening commands will be executed after the authorization has been given. Such communication patterns are quite common in the case study.

In this paper, we will be using a modified version of the PE application. These modifications decouple computational aspects of the original safety-critical components. These computational tasks are amenable to parallelization and aggressive optimization, while satisfying a relaxed set of soft real time constraints. As a result, a safety critical component is split into a non-time-critical and a time-critical task, both of them being certified at the highest levels of safety. On the other hand, several less critical components with no connection to safety critical tasks have been coalesced for didactic reasons.

In the following, we focus on the validation of the hard

[1]This work is supported by the Technological Research Institute (IRT) SystemX, partially funded by the French public program “Investissement d'Avenir”. It is also partially funded by the EMC2 ECSEL and ITEA ASSUME projects.

real time requirements of the time-critical components of the system. The difficulty being that interferences on shared buses and caches of conventional multicores make it impossible in general to establish a practically useful worst-case execution time of a given task [16]. Our goal is to design a software stack and composition methodology enabling hard real-time control code to be isolated from timing interference, while exploiting parallelism among non-time-critical tasks, and still allowing for communications between the two. To solve this apparently paradoxical and infeasible set of constraints, our language provides an automatic inference mechanism for "late" communications between time-criticality levels. Time compositionality may be implemented through mode changes, and introduced incrementally into existing design and validation flows.

II. State of the art

Thread-level parallelism has become unavoidable in any area where performance matters. While specific designs are emerging that combine predictability and performance—e.g. [16]—off-the-shelf multiprocessors designed for mass-market areas are not well suited to timing analysis. Indeed, it is necessary to establish strict bounds on the worst case execution time (WCET) to address the time-predicability requirements of safety-critical systems [19]. If contention of shared resources can not be avoided, the complexity and imprecision of these techniques worsens dramatically. A survey of these researches can be found in a recent paper [12]. Our approach is not to improve timing analyses themselves, but to make those more effective by *controlling how the system is designed, from specification to code generation*. We also hope to reduce the reliance on timing analysis on large parts of its code: *ideally, most of the software components would not need a fully safe worst-case execution time characterization, even though the full system remains globally time-predictable and provably safe*.

We are interested in mass-market commercial off-the-shelf (COTS) platforms, but we believe our approach will also be applied to more predictable classes of multiprocessors, such as the Kalray MPPA [14], [8], increasing resource efficiency. Such extensions are left for future work. Numerous hardware components impact WCET analysis on multiprocessors [7], such as shared caches and shared busses, etc. The main solutions attempt to reduce the general problem to a composition of sequential WCET analyses, enforcing a strict isolation at all levels of the memory hierarchy. For example, software-cache partitioning has been realized for ARM Cortex A9 [18], and other approaches using scratch-pad and multi-ported memories are also possible. Our proposal builds on these ideas to implement spatial and temporal partitioning.

Our proposal was also inspired by the Logical Execution Time (LET) paradigm [15] where the correctness of the system relies on observable input and output times independently of the actual execution time of the system's components. We extend LET with communications across time-criticality partitions, introducing a new protocol for tightly controlling timing isolation. We also leverage the multiple levels of time-criticality in real-world safety-critical applications to relax the timing isolation of parts of the system, improving overall efficiency and reducing certification costs without jeopardizing the safety of the full system. This approach has also been applied to Automotive control applications, but without enforcing hard timing isolation and compositionality [4].

Finally, when compared with the state of the art in mixed-criticality real-time scheduling, our paper proposes a totally different approach. Isolation between low-criticality and high-criticality components is ensured not only temporally, but also functionally, by means of language design and code generation. This approach is fully complementary to the mixed-criticality task models proposed in the real-time scheduling community. As such, it could be a contribution towards aligning academic work on mixed-criticality systems with the notion of mixed criticality introduced in industry standards [11].

III. A mixed-time-critical synchronous language

We designed a simple mixed-time-critical extension of an existing synchronous dataflow language: HEPTAGON [13].[2] It is a research language and compiler with a LUSTRE-like syntax, analogous to the textual language of SCADE SUITE.[3] HEPTAGON features state of the art constructs such as finite state machines and functional arrays with in-place operations. Its compiler implements a clock calculus upon which the generation of efficient embedded code is built,[4] and a number of optimizations to reduce control flow and memory management overhead.

The original PE application has been completely implemented in HEPTAGON, with only low-level I/O and system calls implemented in C. This choice allowed to faithfully implement the original specification, facilitating the application of formal methods or manual certification procedures. Although using mainly a natural language, the specification describes the functions to be implemented in an equational way which suits easily a dataflow language. HEPTAGON has been used to describe the tasks themselves as well as the target-specific code which describes how tasks communicate. We were also able to compile and test the different components early, then proceed with their integration and static scheduling, preserving timing isolation in a compositional way. In later development and validation stages, the tasks have been identified and mapped to specific processors, scheduled and executed in separation, with no functional changes to the program and reusing its high-level communication code. The detailed presentation and discussion of the parallelization and distribution features is out of the scope of the paper (some of the principles can be found in Gérard et al. [5] and

[2]See http://heptagon.gforge.inria.fr for documentation, source code and applications.

[3]http://www.esterel-technologies.com/products/scade-suite.

[4]Clocks can be seen as a type system for sequences of boolean conditions controlling the presence or the absence of values in stream variables, or the stepping of synchronous, stateful nodes in a process network.

Delaval et al. [9]). Instead, we outline the proposed extensions to HEPTAGON, and in particular how to programmatically control what happens when a task misses its deadline. For this purpose, we extend our language with the notion of “tasks” and “punctuality”.

A *task* is defined as a dataflow node which is the smallest partition of the synchronous program after static scheduling and code generation. Tasks may then be amenable to dynamic or time-triggered scheduling. A task is a reactive program, a property inherited from HEPTAGON nodes: it activates repeatedly in response to a signal, its inputs need to be present before it activates (dataflow semantics) and the task’s outputs are present after it terminates its reaction. After the beginning of the task and before its end, the task does not communicate with any other task. It is the programmer’s duty to choose which dataflow nodes in the node instantiation tree will be tasks and thus to define the granularity at which the application is deployed on the target platform.

If ever a task were to be instantiated inside another task, the compiler would simply ignore this information and continue as if the task was a simple node. This does not guarantee time and space isolation across the “embedded” and “embedding” task, but we found it sufficient to ensure time and space isolation across top-level tasks in the system.

Note: the HEPTAGON language allows the programmer to declare “external” nodes and tasks which can be resolved at link time. This allows specific nodes to be written in plain C.

A task is called *unpunctual* if it is not time-critical. In this case, the task can miss its deadline. It may be left to run to completion, ignoring its outputs, or it may be killed or preempted by the system. When it happens, the outputs of the task are not present. To reflect this possibility, we say that the outputs of the task are also unpunctual. We extend the type system of HEPTAGON by assigning a punctuality to each variable and each task. Our language requires that the punctuality of the output variables is always the same as the punctuality of the task. These unpunctual values can be transmitted to other nodes. However, to exploit these values, the programmer must distinguish two cases: either the value has been computed on time or the unpunctual computation timed out and it is not available. The operator `ontime` takes an unpunctual variable argument and returns a boolean which evaluates to true when the variable can be read. In other words, the expression `ontime v` is the clock of the unpunctual variable `v`. This means that the `merge` operator—combining multiple flows with mutually exclusive clocks in HEPTAGON—can be used to build a punctual value by combining the actual value when present and a “default” value otherwise.

In many cases, the “default” value to be used in place of the actual value when a task missed its deadline is a static constant. This happens when there is a value which is always safe. For instance in our case, it is always safe that the component sends no commands or to considers that there is no door correspondence. The programmer can then give a default value to the unpunctual variable which will be used when the actual value is not available. When an unpunctual variable is set to its default value, its clock is the one of the node. (Or the one explicitly declared if it is different.) Sometimes, specifying a default value is not powerful enough. For instance, in Model Predictive Control (MPC), a more suitable reaction may be to repeat the command issued in the last cycle but following a suboptimal, faster prediction [17], [1].

IV. MIXED-TIME-CRITICAL FLOW AND PLATFORM

We review the tool flow and run-time support for the compilation and execution of synchronous programs with multiple levels of time criticality.

A. Compilation

The extensions to HEPTAGON require three main changes.

1) The `task` keyword is a synonym of `node`. It is supported as an annotation in the intermediate representation, which can then be read by the back-end to generate a scheduling table [6] or communication code.
2) The type system is extended with a punctuality property.
3) A transformation pass abstracts the punctuality information such that the resulting program have the same semantics but is lowered to conventional synchronous code without the mixed-time-criticality constructions. Further down, the usual compilation flow can be applied.

We add a “punctuality” attribute to the type of expressions and variables. The type system verifies that variables assigned from an unpunctual task application are also unpunctual and that unpunctual variables may only be passed as argument when the corresponding parameter is also unpunctual. The new construct `ontime` is always typed as boolean.

The additional transformation pass uses clocks to represent the punctuality. Each unpunctual variable is split into two variables: one is a boolean signal representing whether the value has been computed on time or not, the other is the actual value which is only defined on the former clock. The `ontime v` expression for any variable `v` is translated to either `true` if the variable `v` is punctual or to the clock of `v` otherwise.

A task with an unpunctual parameter is split the same way. HEPTAGON allows one parameter to be the clock of another and thus supports this construction.

When a default value is provided for an unpunctual variable, the real value is selected when present, and the default one otherwise; it corresponds to the `merge` construct in HEPTAGON.

The clock variable for each unpunctual variable needs an input from the system telling whether the task generating the encapsulated value has completed on time or not. We introduce for each task call a fresh abstract function which provides this information. It is the responsibility of the back-end to stub these functions appropriately.

It is worth noting that by relaxing the type system to be a bit more lenient on unpunctual arguments we could have allowed interesting constructions, at the cost of additional compilation efforts: since unpunctual tasks are not time-critical, it should be possible to execute them one after another, waiting for the previous task to terminate normally without killing it when

```
node check_command(door_command : command; door_map : int)
   returns (safe_command : command)
let
  safe_command = if door_map <> -1 then door_command else None;
tel

task check_commands(unpunctual door_commands : command^n; door_map : int^n)
   returns (safe_commands : command^n)
let
  if ontime door_commands then
    safe_commands = map<<n>> check_command(door_commands, door_map);
  else
    safe_commands = None^n;
  end
tel
```

Figure 1. Simplified implementation of a safety-critical function which ensures that there are no command for a train or platform door which is not aligned with a corresponding platform or train door, respectively. The `door_map` array provides a valid description of which pairs of doors are aligned, and the `door_commands` is the array of commands computed for each door. The latter is *unpunctual* as it results from a best-effort computation in a mission-critical task; when commands are not computed in time, it is always safe for the passenger that the doors receive no commands.

a deadline is missed. At worst, the unpunctual task missing its deadline could be preempted to ensure it does not use resources (CPU, memory, etc.) which belong to any time-critical task. Thus depending on the criticality of a task, the behavior to follow when a data is not computed on time would not be the same. If the task is time-critical, we use the constructions introduced in this paper to program the temporary transition to a degraded mode. If the task is not time-critical then it can wait until its data are computed. This solution may accumulate delay if multiple tasks miss their deadline or the delay of a task can be compensated by another. For this purpose, we could have allowed unpunctual arguments to be passed to unpunctual tasks without declaring them as unpunctual, but this would involve heavier changes in the type system.

B. Distribution and parallel code generation

At this stage of our research, the code generation procedure is only partially implemented and thus semi-automatic.

Like in LUSTRE, a HEPTAGON program can be seen as a dataflow graph where vertices are operators or instances of other nodes. A program is defined by its root node. All nodes instanciated from a parent task node can be statically scheduled by the HEPTAGON compiler into a single step function. Calls to the node's internal operators and child nodes may even be inlined if desirable.[5] After the program has been compiled, we obtain a dataflow graph whose vertices are tasks and whose edges are communications between these tasks.

In our case study, most dataflow operators are copies of outputs of one task to the input of another, or field selection where a subset of the data is being copied. Other operators, such as arithmetic ones, are also encapsulated into tasks. The punctuality of an encapsulating task is completely determined by the type system which imposes that the operands and the result of an operators must have the same punctuality.

In summary, the first transformation pass produces a dataflow graph where, after inlining and encapsulation, all vertices are tasks and where edges are communications. These tasks must then be allocated and scheduled. Although the HEPTAGON compiler has its own scheduler, it is intended for pure sequential scheduling and for a totally different optimisation goal. Thus, our approach relies on the existence of external tools to allocate and schedule tasks. These tools must take into account the cost of communications as well as the individual execution cost of tasks. For instance, we have implemented a back-end for the LOPHT [3] scheduler: this back-end allow us to feed the scheduler with the list of tasks, data dependencies, clocks and constraints. LOPHT then produces a scheduling table for all these tasks. [6].

Basically, the allocator gives a color to each vertex in the dataflow graph while the scheduler gives a topological order to those vertices. In our framework, allocation just splits punctual and unpunctual tasks. For each computation resource—i.e. for each color in the dataflow graph—one may generate sequential code. This code calls each task in the order given by the scheduler. When the input of a task is produced by another task on the same computation resource, a simple data copy takes place. When the two tasks are allocated in different computation resources, the generated code uses the communication protocol described in the next section.

C. Partitioning and time-triggered communication protocol

Systems with mixed time-criticalities require a strong assurance on the worst case execution time (WCET) of most safety critical tasks. However this is almost impossible to achieve without temporal and spatial partitioning, due to the shared resources of conventional multi-cores [16]. Concretely, concurrent accesses on shared resources, such as a shared L2 cache or a shared scratch-pad memory bank, result in contentions, which prohibit timing analysis at the system level. Building on partitioning techniques, we propose a time-triggered communication protocol to realize contention-avoidance and timing isolation.

A strict spatial partitioning requires independent physical memory and computing units. This is for two reasons: first, to

[5]Recursion is forbidden.

keep the register and memory context of safety-critical tasks away from malware or bugs; second, to avoid concurrent accesses on shared resources, including instruction caches. Such spatial partitioning can be achieved on off-the-shelf hardware platforms, such as the Zynq 7K [23]. It integrates an ARM MPcore (Dual Cortex-A9) and a FPGA permitting asymmetric multiprocessing (AMP) configurations. Each ARM core can run its own software stack, set up separately in the shared DDR or on chip memory (OCM). Precise timers and bus arbitration policies can also be controlled. Moreover, building on the embedded FPGA, a variety of memory and computing units can be implemented, e.g., for communication buffers, and for controlling external I/O. Therefore the time-critical and non-time-critical tasks can be fully isolated on the physical platform. This does come with a significant cost however, as communications and DDR accesses from the time-critical level must be temporally isolated from any other activity. Such temporal isolation may dramatically reduce the effective parallelism available in the program. In particular, to the best of our knowledge, it is not possible to execute two time-predictable tasks in parallel on such a platform.

Our time-triggered communication protocol aims at realizing sufficient temporal partitioning without destroying the potential for parallel execution. This protocol controls not only shared communication buffers by building access time frame and deadline in order to avoid contentions. But it can also tolerate timeout events in order to avoid delaying the time-critical functions by non-time-critical ones. And it achieves this inspite of the presence of communications between the two levels.

In this protocol, each task is split into phases falling in one of six classes:

- time-critical computing (TCCP),
- time-critical copy to buffer (TCTB),
- time-critical copy from buffer (TCFB),
- non-time-critical computing (NTCCP),
- non-time-critical copy to buffer (NTCTB), and
- non-time-critical copy from buffer (NTCFB).

To set the execution deadlines, we rely on a supplied WCET for each of these phases. Fig. 2 shows a sample chronogram of the time-triggered protocol. In this simple example, there are two parallel execution chains, one for safety-time critical tasks, another for mission critical ones, they are executed in parallel on different CPUs.

Phase I: On the time-critical level, at the end of the date $T0$ that is the deadline of previous function, the TCTB function is executed to copy the data to buffer. Its WCET $W0$ is used to define its deadline $T1$ ($T1 = T0 + W0$).

On the non-time-critical level, $T1$ is the deadline for the previous functions. It should be noted that this deadline is not a hard one: late events are tolerated.

Phase II: The NTCFB function of the non-time-critical level transfers the data from the buffer at date $T1$. Then the NTCCP function deals with the data and generates the output results. Finally the NTCTB function transfers the data to the buffer. The sum of WCETs $Y0 + Y1 + Y2$ of NTCFB, NTCCP and NTCTB is used to define the date of $T2$. The temporal gap between $T1$ and $T2$ can be filled by other TCCP functions according to the task scheduling strategy. The WCET value $W1$ should be less than $T2 - T1$.

Phase III: The TCFB function copies the data from the buffer when the data is ready, if no timeout takes place. Otherwise the late event is handled appropriately by the second TCCP task. The maximal WCET value of LCFB and of the backup function defines the deadline date $T3$.

In fact, if missing a deadline on an *unpunctual* communication is harmless, a default value may be enough to react to the timeout event. Otherwise, a more evolved backup function is needed, and its WCET needs to be accounted for at design time, to set up the timeout accordingly. On the other hand, if a timeout should be considered as a fault, the approach is not applicable and the task should not be considered as non-time-critical; fail-safe or degraded mode transitions, or fault-tolerant approaches might be considered.

It should be noted that, not only a communication from non-time-critical to time-critical, but also time-unpredictable *communications* between two time-critical tasks or two parallel time-critical execution chains (operating in parallel on two different CPUs) may be *unpunctual*. This feature makes the protocol suitable for network-on-chip (NoC) systems, such as the Kalray MPPA manycore processor [14]. On such a chip, time-critical tasks can executed in parallel in the different clusters, while implementing *unpunctual* communication between clusters to reduce overhead and certification cost.

V. Validation

As mentioned above, the PE application is coded according to its original functional specification. This specification, written in SysML, details functional and behavioral aspects, risk levels (vital or non-vital), the component architecture, communications and interfaces. We split the vital functions into time critical and non-time critical ones according to their timing requirements, but any atomic behavior is not broken down.

Although the PE application is a reasonably simple use case, it has more than 300 functional and behavioral requirements already: it is a representative industrial SIL (Safety Integrity Level) application. Many other application areas combine the needs for computational components in the loop of safety-critical control. From engine control to monitoring, control engineers need computational tasks in order to improve trajectories, leading to resource optimization, emission reductions, longer maintenance cycles, etc. Model-predictive control (MPC) is one of the best representatives of such computational control applications.[6] Its applicability to real-time systems has so far been limited by its time impredictability [25], making it a prime candidate for mixed-time-critical execution.

Fig. 3 presents a simplified dataflow graph of the PE application. The following safety requirements must be met by time-critical tasks:

[6] `http://www.mpc.berkeley.edu/mpc-course-material`

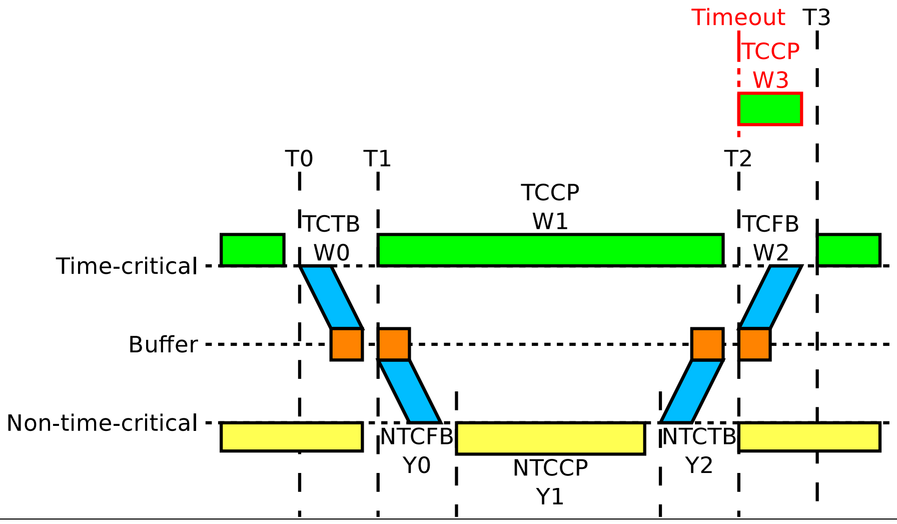


Figure 2. Chronogram of ideal time-triggered communication protocol.

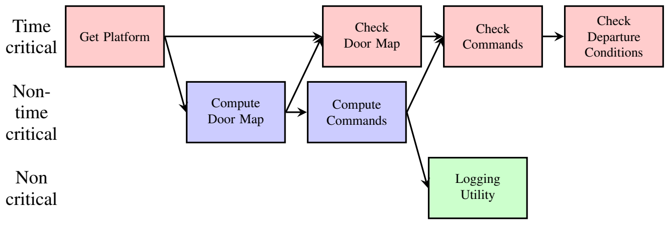


Figure 3. PE application modeling graph.

1) train/platform doors may only open if properly aligned;
2) if the train is not immobilized, doors cannot be opened;
3) doors must be closed to allow the train to depart.

The non-time-critical part sets three other requirements:

1) open/close commands are issued according to the mission;
2) inform passengers of an imminent opening/closing;
3) send warnings to the traffic supervision when the passenger exchange cannot be completed.

These last three requirements are handled by two non-time-critical functions. The first one computes a mapping from train doors to platform doors and vice versa, such that matching doors are physically aligned. The second one uses this mapping to compute commands to be sent to the doors according to the current mission.

To meet the safety requirements, these commands must be checked. If a command breaks one of the requirements, it is canceled: it is always safe, according to these requirements to send no commands. This check relies on three time-critical functions. First, the physical position of the platform doors is retrieved. Then, the door mapping computed by the corresponding mission-critical function is checked using these physical positions and the current train position. Finally, the commands are checked against this mapping and information about the train docking state. The last requirement is ensured by a fourth function which issues a departure authorization when no commands have been sent for the last few seconds.

The application also has a non-critical logging function which records the commands history.

We show in Listing 1 the HEPTAGON source code, declaring two *unpunctual* variables, `door map` and `door commands`. The generated C code is presented in Listing 2. Two functions are generated, for the time-critical code and one for the non-time-critical code, respectively. The communication functions `send` and `receive` have the same parameters: an identifier for the channel buffer, the address and the size of the payload, and the time-criticality of the sender or receiver. The result of the `receive` function is a boolean which indicates whether the communication has been done on time or a timeout has occurred. This boolean can only be false when communicating from a non-time-critcical task to a time-critical one. The communication mechanism is illustrated on a custom system configuration and platform in the next subsection.

A. *Hardware/software implementation*

We selected an off-the-shelf hardware platform, the Zynq 7K SoC ZedBoard [24] as our experimental target. It provides a pair of ARM cores and a FPGA. This flexible platform is very popular in mixed critical execution environments and in hardware-software codesign. In this paper, we leverage the ARM cores for their flexibility and performance on typical software stacks, and rely on the FPGA only for configuring the local memory and bus interfaces, and for implementing communication buffers.

As mentioned in Section IV-C, our first goal is to realize a strict spatial partitioning on the Zynq 7K. This is achieved through asymmetric multi-processing (AMP). The Zynq 7K permits at least two kinds of AMP configurations [22], between two ARM cores and between ARM and FPGA based soft-cores. We selected the ARM core-only AMP configuration to suit our performance-driven implementation strategy:

- ARM core 1 executes the time-critical tasks in a bare-metal environment. The software stack is allocated on

Listing 1. Snippet of the HEPTAGON implementation of the PE.

```
node passenger_exchange(train_position : int)
   returns (safe_door_commands : command^n; departure_authorization : bool)
var
 platform : int;
 unpunctual door_map : int^n;
 safe_door_map : int^n;
 unpunctual door_commands : command^n;
let
 platform = get_platform(train_position);
 door_map = compute_door_map(platform);
 safe_door_map = check_door_map(door_map, platform);
 door_commands = compute_commands(door_map);
 safe_door_commands = check_commands(door_commands, safe_door_map);
 departure_authorization = check_departure_conditions(safe_door_commands);
tel
```

Listing 2. C code generated by the HEPTAGON compiler.

```
void passenger_exchange_tc(int train_position, command safe_door_commands[8],
      bool* departure_authorization) {
 int platform, door_map[8], safe_door_map[8];
 command door_commands[8];
 bool ontime1, ontime2;

 get_platform(train_position, &platform);
 send(0, &platform, sizeof(int), TC);
 ontime1 = receive(1, door_map, sizeof(int) * 8, TC);
 check_door_map(ontime1, door_map, safe_door_map);
 ontime2 = receive(2, door_commands, sizeof(command) * 8, TC);
 check_commands(ontime2, door_commands, safe_door_map, safe_door_commands);
 check_departure_conditions(safe_door_commands, departure_authorization);
}

void passenger_exchange_ntc() {
 int platform, door_map[8];
 command door_commands[8];

 receive(0, &platform, sizeof(int), NTC);
 compute_door_map(platform, door_map);
 send(1, door_map, sizeof(int) * 8, NTC);
 compute_commands(door_map, door_commands);
 send(2, door_commands, sizeof(command) * 8, NTC);
}
```

the on chip memory (OCM) of 256KB (code and data).

- ARM core 0 executes the non-time-critical and non-critical tasks in a Linux environment: Petalinux [20]. The software stack is allocated on the DDR (512MB).
- The communication buffers are implemented on the FPGA.

Figure 4 details the hardware IPs used in the Vivado tool chain [21] for our system configuration.

a) Processing_system7_0: is the IP used to configure a couple of ARM codes. We use almost all default configuration values. It should be noticed that, CPU0 uses both L1 and L2 caches, but CPU1 uses only the L1 cache, in order to avoid concurrent access on the shared L2. The communication buffers on the FPGA are not cached.

b) Proc_sys_reset_0: is the IP used to reset FPGA components controlling by the Processing System.

c) Axi_mem_intercon: is the interconnection IP used to connect the AXI master components with the AXI slave ones. This is a $1 \rightarrow 3$ crossbar as there are one master (processing element) and three slave components: memory block, ZedBoard LEDs and switch GPIO. The latter two interfaces are used during PE application timeout injection and functional test.

d) MEM: is a block memory wrapper containing two IPs, one for the AXI BRAM controller and the other one for a block memory generator. In our implementation, we realize a 32KB memory buffer.

The resource utilization metrics are presented in the table below. Only a small fraction of the FPGA is used.

On the ARM Core1, tasks are time triggered relying on a snoop control unit (SCU) 32 bits local timer. We experimented with two ways to configure the timer: a fixed frequency static value and a dynamic value derived from to executing task's WCET. Both of them were tested successfully.

As the ZedBoard Zynq 7K SoC can operate at 666 MHz, we can set up a static timer reaching a relatively high frequency such as 100 KHz, 500 KHz or even 1 MHz, to accomodate multiperiodic schedules and a wide diversity of task WCET. A "jiffies" counter is used to hold the number of time ticks that have occurred since a task was initiated. At the end of the task,

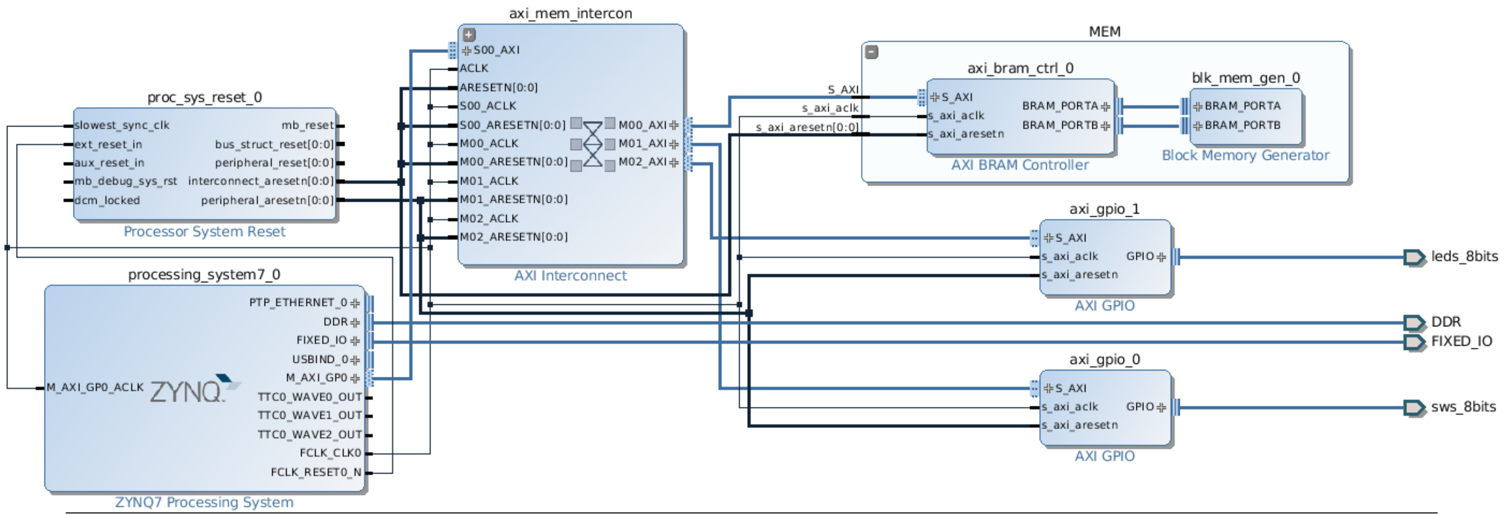


Figure 4. Hardware IPs (Vivado tool chain) used in our implementation.

Resource	Used	Available	Util%
Slice LUTs	2345	53200	4.40%
Slice Registers	2815	106400	2.64%
Block RAM Tile	8	140	5.71%

Figure 5. Mainly resource utilization ratio.

a comparison between the expected WCET and the measured jiffies allows to determine if the deadline was reached.[7] If it did, the next task executes and the jiffies counter is reset to zero. Otherwise, a loop spins until the expected deadline. When using a dynamic timer, it is defined according to the WCET of the operating task, with a "deadline" variable set to detect to the WCET. The rest proceeds identically as with a static counter.

The dynamic scheme requires slightly more code generation effort but it saves CPU resources, saving the need to handle intermediate interrupts. For example in the PE application, the task's largest WCET is $120\times$ the shortest WCET, which means at least 120 timer interrupts can be saved using a dynamic timer.

On the ARM Core0, we did not yet realize time-triggered execution on Linux, but enter a spin loop instead to check if the data is ready. To avoid interferences on the communication buffer due to round robin arbitration, we have to time-isolate the non-time-critical side of the communication as well, which doubles the WCET of the corresponding communication function. This overhead is very lightweight as the communication function has a low cost compared with other components.

The chronogram of the execution is illustrated in Fig. 6;[8] the digits correspond to the following tasks: (1) Get Platform, (2) Compute Door Map, (3) Check Door Map, (4) Compute Commands, (5) Check Commands, (6) Check Departure Conditions, (7) Logging Utility.

As we just presented, the time-critical communication function A runs in parallel with the spin loop, effectively wasting parallel computing resources for a short time period. On the other hand, for most of the execution, functions (3) and (4) run in parallel, showing the benefit of our mixed time-critical design. More complex applications and scalable platforms would make even more effective use of the approach.

Listing 3. C structure of for unpunctual communication.

```
typedef struct communication
{
  volatile int *ready;

  // For unpunctual communication
  volatile int *timeout;

  volatile int *cycle_count;

  volatile void *payload;

  int size;
} communication_t;
```

Listing 3 details the language C structure dedicated to unpunctual communications. This structure contains five fields, *ready* signals that a communication packet is ready, *timeout* informs about missing packets when reaching a timeout, *cycle_count* stores the packet's logical instance cycle (a.k.a. period), *payload* wraps the packet data and *size* represents the size of the payload. We use the "C" and "D" communications of Figure 6 to explain this C implementation.

As shown in the Figure 7, we declare a sender variable C (type of communication_t) on the non-time-critical side, and a receiver variable D (type of communication_t) on the time-critical side.

e) The ready *fields:* of C and D point to a 32 bits physical address allocated on the FPGA Mem Buffer—0x40000000.

[7] We do not attempt to deal with missed deadlines on time-critical tasks, which should be handled as critical faults at another level (e.g., a degraded mode of operation).

[8] The lengths are not on (timing) scale.

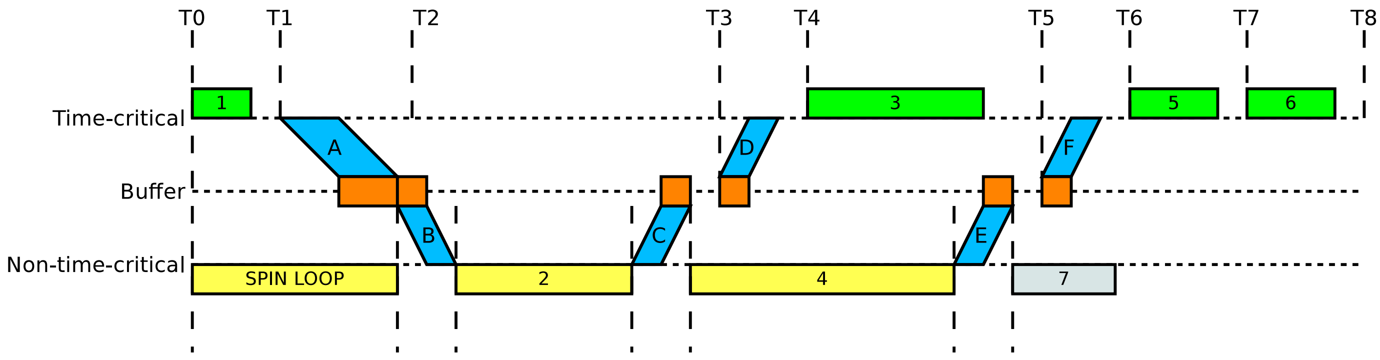


Figure 6. Execution chronogram of PE application.

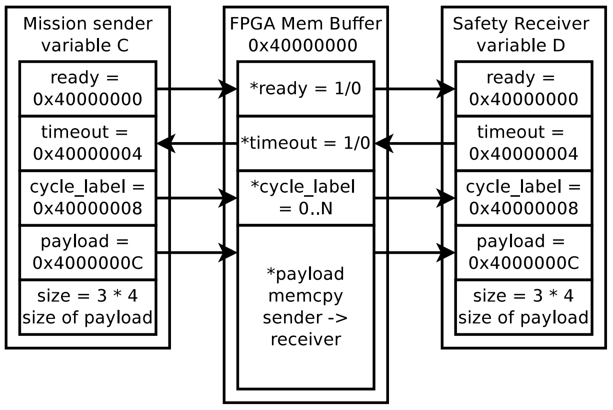


Figure 7. Implementation of the "C" and "D" communication of Figure 6.

The sender variable C produces the *ready* signal, it enables *ready* once the packet *payload* has been copied to the buffer, i.e., when the packet is ready for the receiver. The "ontime" operator is implemented by checking the *ready* field at the receiver side when the deadline date is reached. If *ready* is not enabled, a timeout occurs. The receiver should execute a backup function.

f) The timeout *fields:* of C and D point to the following 32 bits physical address—0x40000004. The receiver variable D produces the *timeout* signal, it enables *timeout* when the deadline date T3 is reached and packet *payload* is not yet ready. That is to say, when the packet timeout occurs. The sender variable C is the consumer; when it seens an enabled *timeout*, it aborts the current packet transmission.

g) The cycle_count *fields:* of C and D point to the next 32 bits physical address—0x40000008. The sender variable C is the producer, the cycle count of the non-time-critical task is sent to the receiver. If it is different from the receivers', a full-cycle delay accumulation occured. That means the sender and the receiver have lost their synchronous association, which is a more severe situation, but still one that the protocol aims to tolerate. It may happen it the non-time-critical task is seriously delayed due to a chaotic accumulation of interferences. In such a case, the simplest approach would be to skip the upcoming instances of the non-time-critical task(s) until the synchrony of the cycle counts can be restored. Optimized methods to handle such severe and accumulated delays are outside the scope of this paper, but the reader interested in advanced strategies may refer to [10].

h) The payload *fields:* of C and D point to the following 32 bits physical address—0x4000000C. To preserve timing isolation, the packet's contents is first copied from the sender to the buffer, then from the buffer to the receiver.

The memory overhead of our time-triggered communication protocol are is limited to the additional communication variables (cycle count, timeout and ready). On the PE application, only 6 such variables are needed to implement the (two) cross-time-criticality communications.

B. Early experiments

Our first experimental validation was to test the mixed-time-critical version of PE application with its original functional self-test, which takes the form of a set of scenario simulations. This test was passed successfully, which proved that the time-triggered protocol does not change the PE function.

Next, we replayed the functional test with an additional timeout injection hazard. We did not change the test scenarios, but inserted controllable delays in the functions (2) Compute Door Map and (4) Compute Commands. Practically, these delays took the form of simple device I/O on the Zync platform, checking if the corresponding ZedBoard switch is on. if yes, a random number of loop iterations were executed in order to simulate a non-predictable delay. If this induces a timeout, the backup function or default values are used, as implemented in the mixed-time-critical synchronous program. The trace of the PE application proved the absence of timing violations or incorrect commands on the time-critical tasks.

As noticed earlier, when a huge delay affects the function (2) or (4), delay accumulation may occur and the *cycle_count* fields of the sender and receiver may differ. Our current implementation simply aborts the PE application by an exception for now, as an illustration of the self-diagnosis potential of the communication protocol. Of course, a complete implementation should skip some upcoming tasks and resynchronize accordingly instead.

VI. Conclusion

We presented an application where mixed criticality resides at the application level, or even at function level, rather than the system level. Moreover, all functions remain safety critical, and the different criticalities we consider are focused on timing predictability and requirements instead. Different time predictability requirements allowed us to expose parallelism and optimize resource utilization, compared to a much more conservative timing isolation of all components. We demonstrate the feasibility of *hard real-time and parallel execution of safety-critical tasks on a conventional embedded multicore platform*. A timing isolation protocol allows best-effort, functionally validated tasks to communicate with hard real-time ones, while (1) *preserving timing compositionality*, and (2) *satisfying all hard real-time constraints* in the complete application. To program and certify such applications, we proposed an extension to a *synchronous language enabling correct-by-construction code generation and parallel execution* on a multiprocessor platform. Based on this experience, we advocate for a multidimensional approach to mixed criticality where *timing constraints are managed separately from other system validation aspects*. We also advocate for a holistic approach, where the design flow of complex control applications takes into account different levels of timing predictability: our proposal is one step towards the construction of such a flow. We encourage control engineers to detail the different admissible modes, *limiting the extent of hard real-time components, and defining the hard real-time reactions to the late availability of non-time-critical data*.

References

[1] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. To appear, 2016.

[2] A. Burns and R. Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[3] T. Carle, D. Potop-Butucaru, Y. Sorel, and D. Lesens. From dataflow specification to multiprocessor partitioned time-triggered real-time implementation. Research Report RR-8109, INRIA, Oct. 2012.

[4] D. Claraz, F. Grimal, T. Leydier, and R. Mader. Introducing multi-core at automotive engine systems. In *ERTS*^{2}, Feb. 2014.

[5] A. Cohen, L. Gérard, and M. Pouzet. Programming parallelism with futures in lustre. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '12, pages 197–206, New York, NY, USA, 2012. ACM.

[6] A. Cohen, V. Perrelle, D. Potop-Butucaru, E. Soubiran, and Z. Zhang. Mixed-criticality in railway systems: A case study on signalling application. In *Workshop on Mixed Criticality for Industrial Systems (WMCIS'2014)*, 2014.

[7] D. Dasari, B. Akesson, V. Nelis, M. A. Awan, and S. M. Petters. Identifying the sources of unpredictability in cots-based multicore systems. In *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, pages 39–48. IEEE, 2013.

[8] B. D. de Dinechin, D. van Amstel, M. Poulhiè, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *DATE invited paper for special session SD1 on Predictable Multi-Core Computing*, pages 24–28, Dresden, Germany, Mar. 2014.

[9] G. Delaval, A. Girault, and M. Pouzet. A type system for the automatic distribution of higher-order synchronous dataflow programs. *SIGPLAN Not.*, 43(7):101–110, June 2008.

[10] J. P. Erickson, N. Kim, and J. H. Anderson. Recovering from overload in multicore mixed-criticality systems. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 775–785, 2015.

[11] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. How realistic is the mixed-criticality real-time system model? In *RNTS*, pages 155–164, Lille, France, Nov. 2015.

[12] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In H. Falk, editor, *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *OpenAccess Series in Informatics (OASIcs)*, pages 31–42, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[13] L. Gérard, A. Guatto, C. Pasteur, and M. Pouzet. A Modular Memory Optimization for Synchronous Data-Flow Languages. Application to Arrays in a Lustre Compiler. In *Languages, Compilers and Tools for Embedded Systems (LCTES'12)*, Beijing, June 12-13 2012. ACM.

[14] Kalray. Kalray MPPA®-256 integrated manycore processor.

[15] C. M. Kirsch and A. Sokolova. *Advances in Real-Time Systems*, chapter The Logical Execution Time Paradigm, pages 103–120. 2011.

[16] A. Pyka, M. Rohde, and S. Uhrig. A real-time capable coherent data cache for multicores. *Concurrency and Computation: Practice and Experience*, 26(6):1342–1354, 2014.

[17] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010.

[18] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson. Making shared caches more predictable on multicore platforms. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 157–167. IEEE, 2013.

[19] R. Wilhelm and J. Reineke. Embedded systems: many cores – many problems. *SIES*, 12:176–180, 2012.

[20] Xilinx. PetaLinux tools.

[21] Xilinx. Vivado design suite.

[22] Xilinx. Xilinx Zynq multi-os support (AMP & hypervisor).

[23] Xilinx. Zynq-7000 all programmable SoC.

[24] ZedBoard.org. ZedBoard.

[25] M. N. Zeilinger, D. M. Raimondo, A. Domahidi, M. Morari, and C. N. Jones. On real-time robust model predictive control. *Automatica*, 50(3):683 – 694, 2014.

DREAMS about reconfiguration and adaptation in avionics

Guy Durrieu* Gerhard Fohler† Gautam Gala† Sylvain Girbal ‡ Daniel Gracia Pérez‡ Eric Noulard*
Claire Pagetti* Simara Pérez Zurita †
*ONERA - France †Technische Universität Kaiserslautern - Germany ‡Thales TRT - France

***Abstract*—The paper describes the reconfiguration approach implemented in the DREAMS middleware to cope with failures and how the concepts are tested on an avionic demonstrator.** [1]

I. Introduction

The DREAMS [Oa13] (Distributed REal-Time Architecture for Mixed Criticality Systems) FP7 project addresses the design of a cross-domain architecture for executing applications of different criticality levels in networked multicore embedded systems.

A. General overview of DREAMS

A DREAMS architecture is composed of several multi-core chips (such as Freescale T4240 [Fre14]) connected through a TTEthernet network [KAGS05]. The DREAMS middleware is in charge of:

1) Ensuring strong temporal and spatial partitioning;
2) Supporting adaptation strategies for mixed-criticality systems to deal with unpredictable environment situations, changes in resource availability, and occurrence of faults;
3) Delivering virtualization technologies for ease of designing.

The DREAMS development methodology and tools are based on model-driven engineering enabling mapping and scheduling of mixed-criticality applications. Three demonstrators, that encompass a broad range of application domains (namely avionics, wind power and healthcare) will be developed and will highlight the DREAMS results.

The project, started in October 2013 with a duration of 4 years, is in its mid-term progress. At this stage, the basic software blocks have been developed and will be integrated next year in the demonstrators.

B. Objective and contributions

This paper focuses on reconfiguration and adaptation strategies and their implementation in the avionic demonstrator. Those strategies only take place upon failures, with the purpose to bring the system back to a safe functioning state. We consider two types of failures:

1) **A permanent core failure**. Intensive integration of small devices on chip increases the *permanent failures* occurrence due to various phenomena such as aging, wear-out or infant mortality [Bor05]. When a core is halted, the partitions executing on the failed core are re-allocated according to pre-computed configurations. We then speak of *reconfiguration*;
2) **A temporal overload situation**, resulting in deadline miss without corrective action. Such a situation may occur because the resources are over-utilized in the nominal mode. However, the timing constraints are respected in degraded modes, that consist in interrupting or degrading the execution of *best-effort applications*. When a *critical application* (i.e. that is not best-effort) detects an internal deadline overrun, the execution moves temporarily the best-effort applications to a degraded mode. We then speak of *adaptation*.

In the following, we describe the resource management proposed in the DREAMS middleware and we define formally the notions of reconfiguration and adaptation (see Section II). We then detail the reconfiguration strategies defined for mitigating the core failures (see Section III) and the adaptation approach for mitigating the temporal overload situations (see Section IV). Finally we give the main ideas of the implementation for the avionic demonstrator and the results obtained by simulation (see Section V). Related works are discussed in Section VI.

II. Resource management in DREAMS

Resource management is a core service provided in the DREAMS middleware for system wide adaptability of mixed criticality applications. The approach is based on the Matrix framework [RF07], but adapted to platforms in which multiple multi-core chips exist and applications can have several criticality levels. Furthermore, the concept of service levels in ACTORS [BBE$^+$11] is extended in DREAMS for its application on virtualized hardware resources instead of applications. The main goals of the integrated resource management are:

- Reconfiguration of a mixed-criticality system upon foreseen and unforeseen changes in its operational and environmental conditions.
- Adaptability mechanisms for securely modifying over the system without interrupting or interfering with its execution.

A. Structure of resource managers

Practically, the resource management services are realized by a Global Resource Manager (GRM) in combination with a set of Local Resource Managers (LRM). The GRM gathers information from the LRMs and provides new configurations

[1]The research leading to these results has received funding from the European FP7-ICT project DREAMS under reference n° 610640.

for the virtualization of resources (e.g., partition scheduling tables or resource budgets). The GRM configuration can include different pre-computed configurations of resources (e.g., time-triggered schedules) or parameter ranges (e.g., resource budgets).

Local resource management services consist of three major parts: Resource Monitors (MONs), Local Resource Schedulers (LRSs) and Local Resource Managers (LRMs). The MON monitors the resource availability and timing of components (e.g., detection of deadline violations). The LRS performs the runtime scheduling of resource requests (e.g., execution of tasks on processor, I/O requests) based on the configuration set by the LRM. The LRM either adopts the configuration from the GRM to particular resources (e.g., processor core, memory, I/O) or selects a new configuration from the ones available and reports state of the resource (from MON) to the GRM.

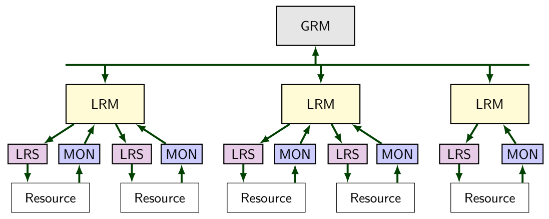


Figure 1. Interaction between resource managers

The LRM and the GRM can be organized in a hierarchical or flat architecture. The flat architecture, shown in Figure 1, consists of a GRM which controls and supervises all LRMs and has a complete view of the system. All LRMs are placed at the same level and they communicate directly to GRM regardless of which resource they monitor or where they are physically located. In the hierachical architecture the LRMs can control underlying LRMs.

B. Implementation choices

DREAMS middleware relies on time and space partitioning principles [Rad05]. In this paper, we consider that those principles are implemented at the chip level by the XtratuM hypervisor [MRC$^+$09], which is a technology involved in the project. Therefore, applications will be executed by a set of partitions. A partition is defined by one or multiple slots, each with a start time and a length. Inside a slot, several tasks can be executed. In the sequel, we will use the color code shown in Figure 2.

Figure 2. Legend

We consider mixed-critical systems where we differentiate two types of application.

Definition 1 (Application model)**.** An application can be a:

- *critical application*. Such an application must respect its timing constraints and in particular the WCET must fit in the allocated slots. Moreover, it cannot be stopped apart if the application encounters an internal error or if the executive layer fails. A critical application app is defined as a set of periodic or sporadic tasks $app = \{\tau_i = (C_i, AET_i, T_i)\}$ where C_i is the WCET, AET_i is the average execution time and T_i is the period or minimal inter-arrival time;
- *best-effort application*. Such an application has less strong constraints. We accept to interrupt them as long as a minimal QoS (quality of service) is ensured. A best-effort application is defined as $app = (U_i, AU_i)$ where U_i is the worst-case asked utilization and AU_i is the average utilization.

A configuration consists in defining temporal slots on the multi-core and mapping the applications in the slots.

Definition 2 (Configuration)**.** A configuration (also denoted *plan* in the hypervisor terminology) consists of:

- a *major cycle* (MaC), the length of which is denoted *MaC_length*;
- a set of slots sl_i distributed over the cores and the MaC. A slot is defined as $sl_i = ([s_i, e_i], n_i)$ where s_i is the start time, e_i is the end time and n_i is the number of core where the slot is allocated;
- a mapping of the jobs of critical applications in the slots. Jobs are unrolled on the MaC and we know for all job $\tau_{i,j}$ in which slot sl_k it belongs to. We know moreover in which order are executed the jobs inside a slot;
- a mapping of best-effort applications in the slots. For instance, app_i is executed in the slots $sl_{j_1}, \ldots, sl_{j_p}$.

C. Definition of the notions of reconfiguration and adaptation

A reconfiguration consists in moving from one configuration to another and this happens when a core has failed. An adaptation consists in degrading a configuration and this occurs when a temporal overload situation happens. Adaptions are handled locally by the LRM whereas core failures may be recovered locally by the LRM or globally by the GRM.

a) Permanent core failures: When a core has failed, the partitions hosted on it are no longer executed. Such a situation can be mitigated by an active redundancy (if some other resource executes the same partitions) or by applying a reconfiguration. Due to the high number of cores provided by a DREAMS platform and the overall resource managements, we decide to incorporate reconfiguration capabilities.

b) Temporal overload situations: The chapter 8 of [But97] focuses on the overload conditions, that are *critical situations in which the computational demand requested by the task set exceeds the time available on the processors, and hence not all tasks can complete within their deadlines*. Such a situation can result for several reasons, e.g. environmental solicitations or fault of peripheral devices or cohabiting applications.

In the DREAMS project, we consider IMA platforms where such problematic situations are usually contained thanks to the temporal isolation. However, we decided to leverage this restriction in order to increase the overall utilization of the multi-core chips. Indeed, we observed that when computing an upper bound of applications WCET on a multi-core chip and reserving this amount of time for all of them leads to an over-provisioning of the platform. As a matter of fact, this WCET is rarely reached and most of the time, the average execution time (AET) is much below the capacity of the platform. This is the reason why we accept a multi-core to be over-utilized by the applications. Our model is detailed in the definition below.

Definition 3 (Under-provisioned platform)**.** Any multi-core can be over-utilized in the following way:

- $\sum_i \frac{C_i}{T_i} + \sum_j U_j > \textit{number of cores}$: the overall utilization exceeds the multi-core capacity;
- $\sum_i \frac{AET_i}{T_i} + \sum AU_j \ll \textit{number of cores}$: the overall average utilization is much below the multi-core capacity;
- $\sum_i \frac{C_i}{T_i} < \textit{number of cores}$: the overall utilization for the critical applications fits the multi-core capacity.

This means that the best-effort applications are those leading to the overtaking of the provisioning. This situation will be handled as proposed in [KPR$^+$14], which means that we will monitor regularly the critical applications and if an internal deadline is exceeded, the best-effort applications will be interrupted and resumed once the critical applications are not any longer endangered.

Note that GRM only makes global reconfiguration decisions when necessary, but it is not required for the continuous operation of the system. The unique failure mode considered for the GRM is the *loss*, due to the permanent failure of the hosting core. Thus, in case of GRM failure, the overall system dependability is not compromised as the system will still keep on executing; just no new global reconfigurations will be possible.

D. Interaction between GRM and LRMs

The interaction between resource management components takes place via Sampling and Queuing ports (provided by the hypervisor). As shown in figure 3, three channels are created between each GRM-LRM pair:

1) Updates channel: For LRM to send resource status updates to the GRM and request for global reconfiguration.
2) Orders channel: For GRM to send reconfiguration messages to the LRM.
3) Membership channel: Each LRM periodically sends a live-signal to the GRM via this channels for the membership purposes.

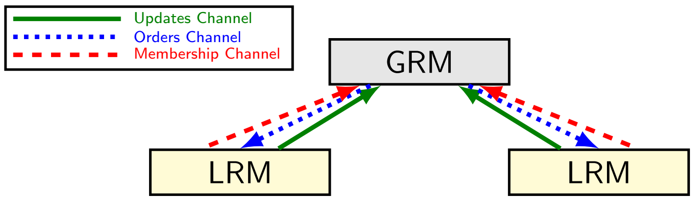


Figure 3. Comunication between RM components

The properties of the RM communication channels are summarized in table I. We remind that TTEthernet protocol [KAGS05] allows several types of traffic:

- rate constraint (RC) traffic: bandwidth guarantee for each application is predefined and delays and temporal deviations have defined limits.
- time triggered (TT) traffic: messages are sent over the network at predefined times and take precedence over all other traffic types.
- best-effort (BE) traffic: it follows the methods of classical Ethernet networks. There is no guarantee whether and when the BE messages can be transmitted, what delays might occur and if they arrive at the recipient.

Table I
SUMMARY OF RESOURCE MANAGEMENT COMMUNICATION CHANNELS

Communication Channel	Port Type	TTEthernet Traffic	Source	Destination
Updates	Queuing	TT	LRM	GRM
Orders	Sampling	TT	GRM	LRM
Membership	Sampling	TT	LRM	GRM

III. Reconfiguration strategy in case of a core failure

When a failure occurs, a detection mechanism must detect the problem and a system recovery procedure must bring the system to a correct state. In the DREAMS project, the detection is based on monitoring (MON) at the multi-core level and recovery at the LRM or GRM level. The recovery procedure is based on pre-defined mode changes executed by the LRM. This entails that a set of possible configurations is computed off-line and that a reconfiguration consists in moving from one configuration to another. The transition steps between mode changes must be safe.

A. Reconfiguration graphs

Since core failures may be recovered locally by the LRM or globally by the GRM, we need to distribute the view of the current configuration between the different stakeholders. The current configuration is represented as the combination of the local configurations. The GRM has an up-to-date system wide vision of the current configuration and stores all the admissible reconfigurations. Each LRM and each switch store a local reconfiguration graph detailing local reconfiguration and global mode changes asked by the GRM.

Definition 4 (Local reconfiguration graphs)**.** A (local) reconfiguration graph is a tuple $\langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ where:

- Q is a finite set of configurations;
- q_0 is the initial configuration;
- $\rightarrow \subseteq Q \times Q$ is the set of local transitions from one configuration to another;
- $\dashrightarrow \subseteq Q \times Q$ is the set of transitions from one configuration to another requested by an other entity.

A reconfiguration graph is stored in each LRM, in each switch and in the GRM. In an LRM, plain arrows represent local decisions while dashed arrows represent decisions provided by the GRM. In the GRM, plain arrows represent local decisions while dashed arrows represent decisions made by some LRM. A switch graph only contains dashed arrows and reconfiguration requests are triggered by the GRM or some LRM.

Since network switch routing tables must be reconfigurable, we must define the reconfiguration strategies for the different types of traffic:

- For rate constraint (RC) traffic, the VLs are defined with their BAG and maximal packet size. Therefore, if an application is reconfigured on the same multi-core by a local reconfiguration then it has no impact on the routing table. If the reconfiguration is global, then several routing tables must be pre defined.
- For time triggered (TT) traffic, it depends whether the instant of emission of packets is related to the offset of the partition slot. If not, then the same reasoning as for RC traffic applies. Otherwise, we must consider the link between offsets of local reconfigurations and network TT messages scheduling.
- For best-effort (BE) traffic, the same reasoning as RC traffic applies.

The GRM stores the complete view of the system which is represented as a global reconfiguration graph.

Definition 5 (Global reconfiguration graph)**.** A global reconfiguration graph is a tuple $\langle Q, \rightarrow, \dashrightarrow, q_0\rangle$ which consists of the product of all local reconfiguration graphs $\langle Q^i, \rightarrow^i, \dashrightarrow^i, q_0^i\rangle$ from the LRMs and the switches together with the GRM local graph $\langle Q^G, \rightarrow^G, \dashrightarrow^G, q_0^G\rangle$. More precisely:

- $Q = Q^1 \times \ldots \times Q^n \times Q^G$,
- $q_0 = (q_0^1, \ldots, q_0^n, q_0^G)$,
- $\rightarrow \subseteq Q \times Q$ is defined as

$$((q^1, \ldots, q^n, q^G), (p^1, \ldots, p^n, p^G)) \in \rightarrow$$
$$\Longleftrightarrow$$
$$\begin{cases} \exists i \in \{1, \ldots, n\}, \\ \qquad (q^i, p^i) \in \dashrightarrow^i \wedge \forall j \neq i, q^j = p^j \wedge q^G = p^G \\ or\ (q^G, p^G) \in \rightarrow^G \wedge \forall j, q^j = p^j \end{cases}$$

- $\dashrightarrow \subseteq Q \times Q$ is defined in a similar way than $\rightarrow$ by replacing $\dashrightarrow^i$ with $\rightarrow^i$ and $\rightarrow^G$ with $\dashrightarrow^G$.

Figure 4 illustrates the reconfiguration graphs stored by the different resource managers and switches. The chip on the right hand side has several pre-defined configurations named from C1 to C7. The switch on the right hand side has several pre-defined configurations named from S1 to S3. For the GRM, we only show the global reconfiguration graph as the product of all local reconfiguration graphs.

B. Local vs. global decisions

Example 1 (of local reconfiguration)**.** Failure f1 (a core halt) occurs in the multi-core T1. According to the reconfiguration graph of T1, the LRM will move to configuration C2. A message must be sent to the GRM so the latter can maintain an updated configuration. This is shown in Figure 5.

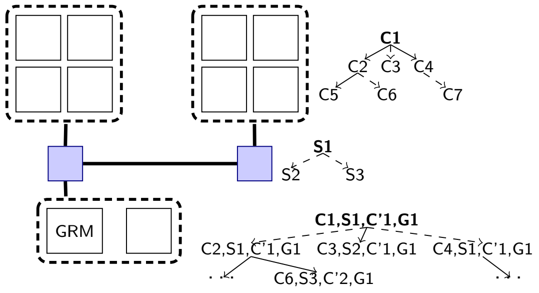


Figure 4. Distributed reconfiguration graphs

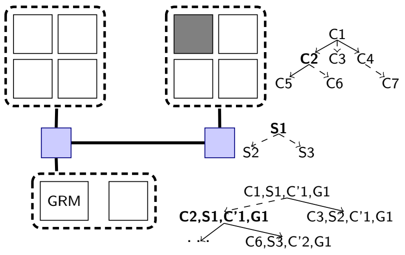


Figure 5. Local reconfiguration

Example 2 (of global reconfiguration)**.** Failure f5 (a core halt) occurs in the multi-core T1. According to the reconfiguration graph of T1, the LRM has no solution. Thus it informs the GRM. The GRM can apply a global reconfiguration: applications running on the failed core of T1 will be reconfigured in T2. The GRM informs (1) T2 to load and execute the applications, (2) T1 that a reconfiguration is applied, (3) the switches to reconfigure the routing tables (messages are emitted by T2 and not T1). An ack by T2 may be expected. This is shown in Figure 6.

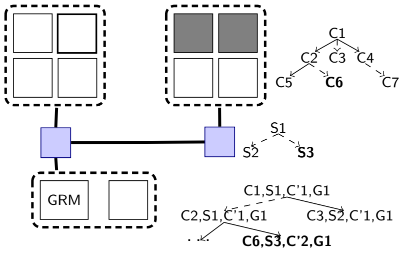


Figure 6. Global reconfiguration

C. Detailed specification

In this section, we explain how the resource management services are implemented at the chip level.

1) MON: executes a service regularly in each core to detect the core's health. If the core is working correctly, the service writes to a shared structure that everything is fine. Otherwise, if the core has failed, the service is not activated and is not able to update the shared structure.

The cores update asynchronously the structure at distinct pre-defined times and check the other cores status at that

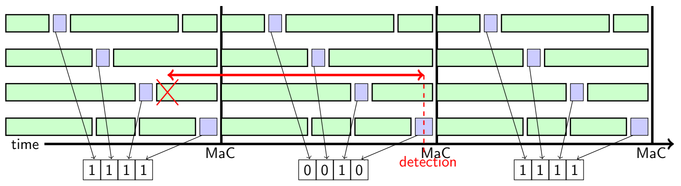


Figure 7. Example of core failure detection on a quad-core.

moment. For example let us consider a quad-core where the MON service is executed in each core only once per major cycle (MaC), see Figure 7. Let us suppose that one of the cores fails just after the MON execution (represented as a red cross on core 3). The detection will be done by the core 4 in the next MaC (the time needed for the detection is shown as a red arrow).

2) LRM: Once a core failure has been detected by the MON, the latter informs the LRM. The time between the detection by the MON and the execution of the LRM has a direct influence on the response time for reconfiguration. This is the reason why we impose the MON and LRM to have pre-defined slots next to each other in order to minimize the delay between the detection and decision. Figure 8 gives an example of a decision of the LRM after the detection of the failure of core 2.

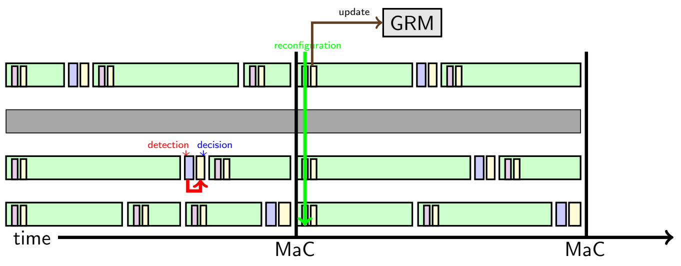


Figure 8. Example of LRM decision after a core failure detection.

Once the LRM is informed by the MON service of a core failure, it has two possibilities:

- a local reconfiguration is possible according to its local reconfiguration. In that case, it asks the LRS to change the plan at the end of the MaC for the new configuration one. The reallocated partitions are re-started in a default state, no context has been stored from the previous executions. The unchanged partitions continue their execution transparently. This transition step is then safe;
- no local reconfiguration can recover from the situation. In that case, the critical tasks are locally reconfigured in priority if possible (pre-computed configuration) while some best-effort applications may be removed. Then, the LRM informs the GRM that some applications cannot be hosted any longer on the multi-core platform and it is up to the GRM to find a global reconfiguration.

The DREAMS project requirements state that a critical application cannot split onto different cores of a multi-core. Thus, when a global reconfiguration must be taken, complete applications are thus reconfigured on different cores. A future work could consider to parallelize the applicative code onto different cores, but at the price of modifying the applicative code.

3) LRS: The LRS is more detailed in section IV-B3, because it plays a more important role for the temporal overload situations. The LRS is in charge of scheduling the tasks inside the slots. For the core failure case, it just reads the current configuration and applies it.

IV. Adaptation strategy in case of a temporal overload situations

In the DREAMS project, we under-provision the platform to increase the average performance. Such an approach can in some cases lead to problematic situations where critical applications may overrun their deadlines. To forbid this timing failures, a detection mechanism is in charge of analyzing intermediate deadlines and adapt the processor demands by interrupting the best-effort applications.

A. Adaptation tables

An adaption consists simply in interrupting the best-effort applications. It is therefore sufficient to store statically the partition identifier of the best-effort applications. Since the adaptation mechanisms are combined with the reconfiguration capabilities due to the core failures management, those identifiers must be stored for all reachable configurations.

Definition 6 (Adaptation table)**.** An adaptation table consists, for each configuration defined in the reconfiguration graph, of a list of applications.

B. Detailed specification

In this section, we explain how the resource management services are implemented at the chip level.

1) MON: extends the deadline warning detection method described in [KPR$^+$14]. In this initial work, only standard tasks sets were considered and the schedule consisted in executing a task alone on a core. In the DREAMS project, we consider partitions slots and the MON/LRM/LRS components. The idea is that each critical application monitors its execution and checks if the application is in danger of overrunning its deadline. If it is the case, then the MON service signals to the LRM that a deadline overrun will probably occur.

The partition slots for critical applications contain internal *observation points* which are defined off-line and correspond to the moments where the MON is executed. We choose to monitor the temporal behaviour between tasks in the slot. This way we do not modify the partition code. This illustrated in Figure 9.

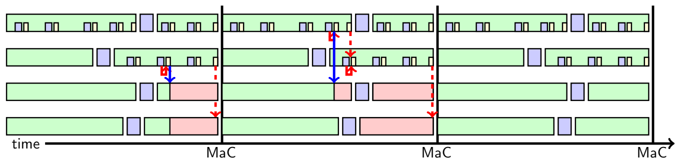


Figure 9. Example of internal deadline failure adaptation when LRM inside critical tasks.

The monitoring checks if the interferences of the low criticality tasks can be tolerated by verifying a safety condition. The safety condition in the initial work [KPR+14] consisted in checking that in the next observation point we would still have time to switch to the *degraded mode* (or isolated mode, in the sense that only critical applications may run). This required numerous information, such as the remaining WCET of the partition *Part* in isolated execution from the observation point x until the end. Thanks to the positioning of observation points between tasks, the safety condition can drastically be simplified as shown in Eq. 1.

$$\mathsf{ET}(x) \leq \textit{internal deadline}(x) \qquad (1)$$

where $\mathsf{ET}(x)$ is the monitored execution time of *Part* until point x and *internal deadline*(x) is a pre-computed constant giving the maximal possible internal deadline.

2) LRM: stores the adaptation graphs and knows which applications must be suspended. This action is immediate (compare to the reconfiguration which occurs at the next MaC). Suspended applications are re-started once all running critical tasks have not asked to move to the degraded mode.

3) LRS: The LRS starts its execution as soon as a partition slot starts. The first time the LRS is executed (typically during plan 0 schedule of the hypervisor) it launches the application initialization, which sets up its internal state for execution. After that, the LRS initializes the application tasks schedule during the different slots and plan configuration.

Afterwards during the major cycles the LRS is executed at the beginning of each slot and it launches a predefined and sequential list of partition/application tasks for that slot, and once all the tasks have been executed the LRS stops its execution, even if time remains in the current partition slot. Note that an LRS execution can span multiple partition slots, but to facilitate the LRS for critical partitions comprehension we will always suppose that a LRS execution starts and finishes in the same partition slot.

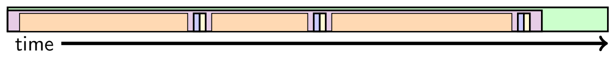


Figure 10. Example of critical partition slot execution.

Figure 10 shows an example of critical partition slot execution under the control of the LRS. In between the execution of two tasks the MON and the LRM are executed to:

- the MON execution collects the performance monitors of the just executed task and the current execution time of the slot,
- the LRM execution determines if an adaptation is needed.

V. Avionic demonstrator

The DREAMS architecture avionic demonstrator will highlight the reconfiguration capabilities of the middleware. The demonstrator combines critical applications with non-critical applications using heterogeneous multi-core platforms, connected using a wired network.

A. Applications involved in the demonstrator

Figure 11 shows the five applications/functions deployed in the avionics demonstrator, three critical ones and two non-critical. The critical applications are: (1) a Flight Management System (FMS, previously described in [DFG+14]), (2) a Display Management System (DMS), and (3) a Sensors Data Provider (SDP). The non-critical applications are: (1) an In-Flight Entertainment (IFE), and (2) the panels.

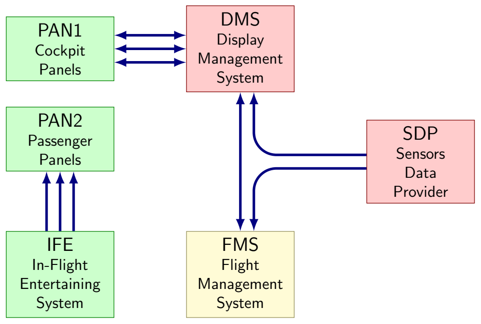


Figure 11. Inter-function communication in the avionic use-case

The FMS aims at performing in-flight guidance of aircrafts, which is based on the use of *flight plans* selected before departure, either by the pilot or a dispatcher for airliners. A flight plan includes basic information such as departure and arrival points, in-flight waypoints, estimated time en route, alternate landing airports, expected weather conditions, and so on. The SDP is the application responsible of collecting the sensors signals such as the GPS or the anemo-barometric probes. The SDP packetizes the sensors data and sends them to the FMS and the DMS. The DMS manages the information to be displayed on the cockpit panels for the pilots. The DMS also takes care of sending pilots commands to the FMS. While the cockpit panels are typically highly critical applications, their study is out of the scope of the DREAMS avionics demonstrator, so they are considered as non-critical. The IFE is connected to the passenger panels to broadcast video streams.

Table II
Avionic use case specifications

Function	DAL	Max unavailability	Number of tasks (periodic, aperiodic)
FMS	B	600ms	26 (10, 16)
SDP	A	600ms	5 (4, 1)
DMS	A	1000ms	7 (6, 1)
IFE	E	∞	NC
Panels	E	∞	NC

Table II summarizes the specification of each function in terms of criticality level (DAL), Maximal Unavailability, and a brief task set description. The Maximal Unvailabilty corresponds to the maximum allowed time to perform reconfiguration (maximum suspended time for the task due to a failure). The In-Flight Entertainment and the panels are implemented in commodity computers with standard OS (i.e., Linux) or in non-critical partitions, and as such the task description is not considered (NC) in this study.

B. Demonstrator platform

Three different computing platforms are used for the deployment of the different applications:

- **Freescale T4240** [Fre14] (see Figure 13): The T4240 is a 64bit PPC architecture with 12 cores organized in 3 clusters of 4 cores interconnected connected through a propietary NoC to 3 different memory controllers, each one with a dedicated L3 memory cache. A PCIe TTEthernet card is also attached to the T4240 to satisfy the network requirements.

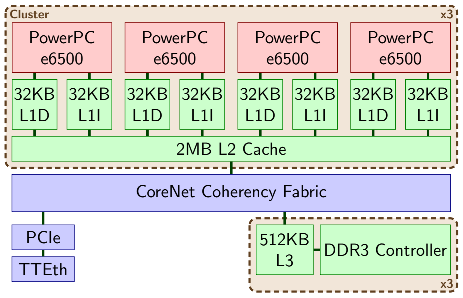


Figure 13. PowerPC T4240 architecture

- **DREAMS Harmonized Platform** [Oa13]: The Harmonized Platform is a development board with Xilinx Zynq-7000 SoC containing ARM Cortex-A9 cores and an FPGA. The DREAMS hardware solution like the Spidergon NoC, enhanced NoC interfaces and TTEthernet controller are implemented on the FPGA. Additionally, three Microblaze soft processor cores are connected to the NoC. The DDR memory controller can also be accessed by all the computing resources via the NoC.
- Regular PC: Regular PCs are used to deploy non-critical applications like the panels.

The applications are run on top of the XtratuM hypervisor enhanced with the DREAMS solutions, i.e., the MON, LRM and the GRM among others. Figure 12 shows one of the targeted deployments of the applications over the aforementioned hardware platforms.

The communications between the different computing platforms are ensured by: two TTEthernet switches, two PCIe TTEthernet cards and the TTEthernet controller embedded in the DREAMS Harmonized Platform. Regular ethernet cards are used in the PCs, as the applications on those systems don't require any safety level communication (i.e., time triggered or rate constrained). The demonstrator mixes the three types of traffic supported by TTEthernet:

- best effort for the communication from/to non-critical applications,
- and time triggered and/or rate constrained for the communication between the critical applications and the communication between the DREAMS services, as the communication between GRM and LRMs.

C. Results

Currently, the fault tolerance mechanisms have been implemented in XTRATUM. But the hypervisor has not yet been ported on the T4240. Therefore, we could not run experiments on the avionic demonstrator. Instead, we made several simulations and prepared a series of fault-injection scenarios to validate the approach.

1) Reconfiguration graph and adaptation table: The initial configuration q_0 in the T4240QS of the left hand side of Figure 12 is defined as $\langle MaC\_length = 200ms, \{sl_i\}, alloc\rangle$ as shown in Figure 14 whereas TTE stands the TTEthernet driver.

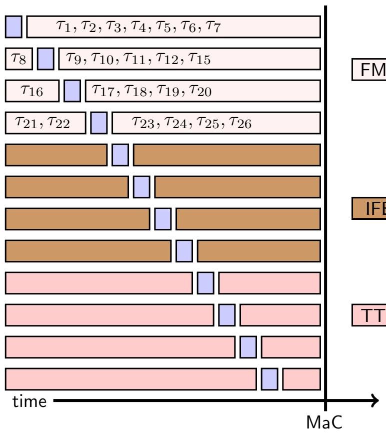

$\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7$
τ_8
$\tau_9, \tau_{10}, \tau_{11}, \tau_{12}, \tau_{15}$
τ_{16}
$\tau_{17}, \tau_{18}, \tau_{19}, \tau_{20}$
τ_{21}, τ_{22}
$\tau_{23}, \tau_{24}, \tau_{25}, \tau_{26}$
FMS
IFE
TTE
time
MaC

Figure 14. Initial configuration

The adaptation table for this configuration is given by

```
const int adaptation_point[NB_PARTITION][NB_MAX_POINT]={
  {20,  45, 100, 130, 170, 190},
  /* partition 0: 6 observation points,
     max time to reach point 1 = 20 */
  {-1,  -1,  -1,  -1,  -1,  -1},
  /* partition 1: 0 observation point */
  {40,  70,   90, 130,  -1,  -1},
  {-1,  -1,   -1,  -1,  -1,  -1},
  {80, 110,  150,  -1,  -1,  -1},
  {20,  -1,   -1,  -1,  -1,  -1},
  {90, 120,  180,  -1,  -1,  -1}
};
```

2) Fault-injection scenario: We have defined some scenarios that will be used to evaluate the DREAMS Local Resource Management services. The target architecture in all the scenarios will be based on the avionic demonstrator.

a) Scenario of double core failures: The purpose of the scenario is to test the local and global reconfiguration capabilities. The scenario is similar to the one detailed in examples 1 and 2 of Section III-B. Two faults are injected: (1) a core fails on the multi-core 2 leading to a local reconfiguration; (2) a second core fails on the same multi-core leading to a global reconfiguration whereas DMS is reconfigured on the second multi-core. To inject the fault, the MON will be modified not to update the share structure. The objective of this scenario is to:

- check that the LRM adaptation capabilities do not affect the safety of critical applications which were not hosted on failed cores;

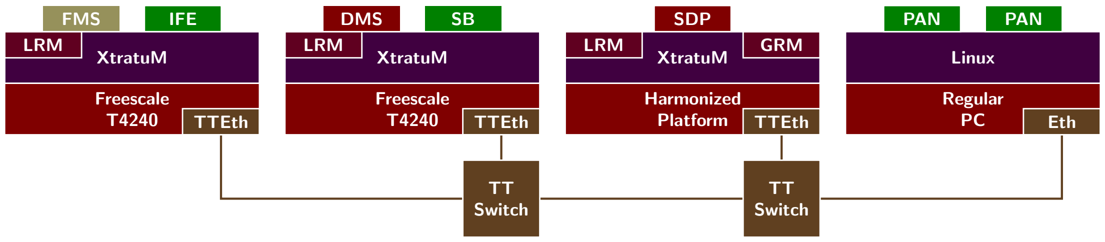


Figure 12. Avionic demonstrator architecture

- determine the gain due to local reconfigurations versus global ones;
- compare several timing parameters to improve the reconfiguration response times.

b) Scenario of temporal overload situation: The second scenario aims at testing the LRM capacity to interrupt low criticality tasks. The scenario consists in (1) detecting a deadline overrun in the FMS partition of multi-core 1; (2) interrupting the IFE execution after the detection until the end of the FMS slot. The objective of this scenario is to:

- check that the LRM adaptation capabilities maintain the predictability of the safety of critical applications;
- compare several timing parameters to assess the interruption response times and variability of the execution time of the safety critical applications.

3) Simulation: Since XTRATUM is not ported yet on the T4240QS, we run the scenarios with the QEMU simulator. The observed behaviours were those expected. In the next months, we will port the work on the real target.

VI. Related work

Reconfiguration for avionic platform has been proposed in Asaac [ASA04] project for military aircrafts; Diana [EJS+10] and Scarlett [PBB+12] for the civil domain. In all cases, reconfigurable IMA was able to change the configuration of the platform by moving applications hosted on a faulty computing module to spare computing modules. The main objective of such an extension was to reduce the cost of unscheduled maintenance and to improve the operational reliability of the aircraft while preserving current safety levels. In Diana the approach was distributed while in Scarlett the reconfiguration was centralized. In DREAMS, a module is based on a multi-core architecture and failures depend on this new hardware, and while the global reconfiguration of the system is centralized like in Scarlett the modules can perform local reconfigurations when one of the cores fails.

a) Permanent failures: The standard approach to deal with permanent failures is based on replicating applications and components into multiple copies. Such redundancy can be achieved via hardware or software mechanisms. Since the avionic demonstrator relies on COTS multi-core, only the second case could be considered. Most existing approaches target anomalous behaviours. For instance, the authors of [SHLR+09] present the *symptom based detection and diagnosis* principle, developed during the SWAT (SoftWare Anomaly Treatment) project, to manage faults in multi-core architectures running multi-threaded software. For permanent fault, the detection algorithm is based on a deterministic replay to diagnose the faulty core and the run-time is in charge of isolating the failed core.

The authors of [GLSS01] describe a fault-tolerant scheduling approach to support permanent core failures but they do not target a real software implementation.

b) Temporal overload situation: The objective of run-time monitoring is to check on-line, during the real execution, the timing behaviors of the system and verify if they are compliant with an abstract view of the expected behavior. If the system diverges from the specification, then a recovery may be applied.

In [BLS06] and [RRF10], the timing specifications are expressed with Timed Linear Temporal Logic (TLTL) formulas. Practically, timed automata are used to implement the valid behaviors and the decision layer stores the automaton description including the location invariants and the transition table. The verification function works for each event as follows: either it is valid and the execution continues or the event is invalid, in which case a recovery procedure or an error is called. Such an implementation requires a strong synchronization between the application and the monitor, and in particular it is necessary to ensure mutual exclusion. In [BFR13], the monitoring strategy has been extended for multi-threaded code. The monitor is decomposed into pieces that apply local detection while exchanging messages to ensure a coherent checking.

In the automotive domain, tasks can exhibit a dynamic real-time behavior, e.g. the period depends on the engine speed. This variability leads to a continuous change in the configurations taken into account by the OS schedule on the processors. We then speak of multi-mode applications that can switch between different operational modes at run-time. Such a change of rate may make the system unschedulable and the engine control inefficient or even unstable. The authors of [NES12] propose mode changes without violating timing constraint by pre-computating the possible behaviours. The approach consists in analyzing all potential run-time scenarios and study in details the critical ones. The possible recoveries are the following: degraded functional execution (with restricted WCET), abort or suspend low criticality tasks.

Several approaches propose resources reallocation based on information derived from monitoring their utilization, e.g. the memory accesses. For instance, in [NPB+14] interference-sensitive WCETs are computed based on a preliminary analysis of the resource usage of tasks. The shared resources are off-line partitioned among tasks. A run-time monitoring device observes the resource usage of each task and suspends the task that overtakes the allocated capacity. In [NP13] the approach is extended by allowing safe dynamic changes in the resource partitioning, when resources are underutilized. In [YYP+13] an approach has been developed to reserve memory accesses for critical tasks. A run-time controller has been implemented which regulates the accesses to the shared memory and ensures temporal isolation among tasks. An off-line profiling technique has been proposed in [MDB+13] which finds the most frequently accessed memory pages in a task. Then, this information is used to modify the variables position in the shared caches in order to reduce the interferences.

VII. Conclusion

We have described the main ideas of the reconfiguration and adaptations strategies proposed in the DREAMS middleware. In the next year, the building blocks will be ported on the avionic demonstrator and the fault injection scenarios will be run on it.

The implemented monitoring techniques are simple and can be improved by defining an adapted set of rules depending on the current configuration. This is the idea developed in [GPBB08] where a safety mode automaton is constructed from the system and the environment. We will study how such ideas could be applied to increase the quality of the reconfiguration. Concerning the adaptation, the QoS for the best-effort applications has not been investigated yet and this will be also an axis of future work.

Finally, this article has addressed the technical aspects of the reconfiguration and adaptation strategies. However, in order to be applied to industrial solutions, the certifiability of the approach requires further study.

References

[ASA04] ASAAC. ASAAC final draft of proposed guidelines for system issues - volaume 4 : System configuration and reconfiguration, 2004. Aeronautical Radio INC.

[BBE+11] Enrico Bini, Giorgio C. Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Karl-Erik Årzén, Vanessa Romero, and Claudio Scordino. Resource management on multicore systems: The ACTORS approach. *IEEE Micro*, 31(3):72–81, 2011.

[BFR13] Olivier Baldellon, Jean-Charles Fabre, and Matthieu Roy. Minotor: Monitoring timing and behavioral properties for dependable distributed systems. In *19th Pacific Rim International Symposium on Dependable Computing, PRDC'13*, pages 206–215, 2013.

[BLS06] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337, pages 260–272, 2006.

[Bor05] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, November 2005.

[But97] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, volume 24 of *Real-Time Systems Series*. Springer, 1997.

[DFG+14] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Proceedings of the 7th Conference on Embedded Real Time Software and Systems (ERTS'14)*, 2014.

[EJS+10] Christian Engel, Eric Jenn, Peter H. Schmitt, Rodrigo Coutinho, and Tobias Schoofs. Enhanced dispatchability of aircrafts using multi-static configurations. In *Embedded Real Time Software and Systems Congress (ERTS 2010)*, Toulouse, France, 2010.

[Fre14] Freescale. T4240 QorIQ: Integrated multicore communications processor family reference manual, 2014.

[GLSS01] Alain Girault, Christophe Lavarenne, Mihaela Sighireanu, and Yves Sorel. Generation of fault-tolerant static scheduling for real-time distributed embedded systems with multi-point links. In *15th International Parallel & Distributed Processing Symposium (IPDPS-01)*, page 125, 2001.

[GPBB08] Jérémie Guiochet, David Powell, Etienne Baudin, and Jean-Paul Blanquart. Online Safety Monitoring Using Safety Modes. In *Workshop on Technical Challenges for Dependable Robots in Human Environments*, pages 1–13, May 2008.

[KAGS05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered ethernet (TTE) design. In *8th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pages 22–33, 2005.

[KPR+14] Angeliki Kritikakou, Claire Pagetti, Christine Rochange, Matthieu Roy, Madeleine Faugère, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems. In *Proceedings of the 22th International Conference on Real-Time and Network Systems (RTNS'14)*, pages 139–148, 2014.

[MDB+13] Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-time cache management framework for multi-core architectures. In *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*, pages 45–54, 2013.

[MRC+09] Miguel Masmano, Ismael Ripoll, Alfons Crespo, J.J. Metge, and Paul Arberet. Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[NES12] Mircea Negrean, Rolf Ernst, , and Simon Schliecker. Mastering timing challenges for the design of multi-mode applications on multi-core real-time embedded systems. In *Proceedings of the 6th Conference on Embedded Real Time Software and Systems (ERTS'12)*, 2012.

[NP13] Jan Nowotsch and Michael Paulitsch. Quality of service capabilities for hard real-time applications on multi-core processors. In *21st International Conference on Real-Time Networks and Systems (RTNS'13)*, pages 151–160, 2013.

[NPB+14] Jan Nowotsch, Michael Paulitsch, Daniel Bühler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *26th Euromicro Conference on Real-Time Systems (ECRTS'14)*, 2014.

[Oa13] Roman Obermaisser and al. DREAMS: Distributed REal-time Architecture for Mixed Criticality Systems. http://dreams-project.eu, 2013.

[PBB+12] Claire Pagetti, Pierre Bieber, Julien Brunel, Kushal Gupta, Eric Noulard, Thierry Planche, Francois Vialard, Clément Ketchedji, Bernard Bésinet, and Philippe Despres. Reconfigurable ima platform: from safety assessment to test scenarios on the scarlett demonstrator. In *Proceedings of the 6th Conference on Embedded Real Time Software and Systems and Software (ERTS'12)*, 2012.

[Rad05] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (ima) development guidance and certification considerations, 2005.

[RF07] Larisa Rizvanovic and Gerhard Fohler. The matrix - a framework for real-time resource management for video streaming in networks of heterogenous devices. In *The International Conference on Consumer Electronics 2007*, January 2007.

[RRF10] Thomas Robert, Matthieu Roy, and Jean-Charles Fabre. Early Error Detection for Fault Tolerance Strategies. In *18th International Conference on Real-Time and Network Systems (RTNS'10)*, pages 159–168, Toulouse, France, 2010.

[SHLR+09] Siva Kumar Sastry Hari, Man-Lap Li, Pradeep Ramachandran, Byn Choi, and Sarita V. Adve. mSWAT: Low-cost Hardware Fault Detection and Diagnosis for Multicore Systems. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 122–132, 2009.

[YYP+13] Heechul Yun, Gang Yao, R. Pellizzoni, M. Caccamo, and Lui Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *19th Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*, pages 55–64, 2013.

A Multi-Core Interference-Aware Schedulability Test for IMA Systems, as a Guide for SW/HW Integration

Soukayna M'Sirdi[1,2], Wenceslas Godard[1], and Marc Pantel[2]

[1]Airbus Group Innovations, Toulouse, France, surname.lastname@airbus.com
[2]IRIT, Toulouse, France, surname.lastname@irit.fr

***Abstract*—In this paper we propose a framework for the automated integration and timing analysis of IMA (Integrated Modular Avionics) applications on multi-core environments. To do so, we present a derivation of the response time analysis formulation by Kim et al. in [12] that takes into account inter-task interference due to sharing the access to the main memory. We adapt the work in [12] to propose a sufficient schedulability test that is adapted both to IMA systems and heterogeneous multi-core platforms. We then exploit this test to guide the design space exploration during the SW/HW integration phase, to select a partition-to-core allocation so that all deadlines are met despite the existence of hardware interference.**

Keywords: multi-core – IMA – interference – response time analysis

I. Introduction

A. IMA Applications

Modern avionic software systems are designed according to the Integrated Modular Avionics (IMA) architecture model, where (i) applications are defined as one or several software partitions; (ii) several partitions can share the same hardware resources, provided that constraints on time and space isolation are respected between each partition. Such isolation requirements are imposed for safety reasons, to prevent the propagation of a fault that happened inside a specific partition, to all other partitions in the software embedded in a system.

IMA software is organized in two layers: at the top level, a schedule is set in advance to plan the executions of the partitions in a TDMA-like fashion. To do so, activation offsets and time windows are allocated to each partition. We refer to this top level schedule as the global schedule of the system. Within the boundaries of the time window of a given partition, a local scheduler dynamically executes the software tasks inside each partition, usually according to fixed priority preemptive policy. The local scheduler operates with no knowledge about the global schedule. In the other hand, the global schedule has to suit the tasks needs, in the sense that the sizes of the partitions time windows must adapt to the tasks running during these time interval. In this paper, we compute the size of a partition's time window such that the task with the lowest priority level only executes once per window. How we compute the activation offsets of the time windows is out of the scope of this paper though.

During the integration phase of a system design, the allocation of the software platform onto the hardware platform is done, as well as the generation of a valid static global schedule. Setting in advance the global schedule enables to enforce an execution plan that has been verified and validated first. In particular, the global schedule must fit the needs of the local schedule inside the time window of each partition, and always ensure the respect of every deadline defined in the entire system. To verify and validate the global schedule, a formal proof of correctness, such as with static timing analysis, is the preferred choice for certification authorities.

B. Multi-Cores and Inter-Task Interference

In multi-core processors, all cores have simultaneous access to shared resources, through shared interconnections. All the requests made to one resource cannot be processed at the same time, which results in waiting delays at runtime. Such interference between tasks of different partitions, simultaneously executing on different cores, significantly extend the execution times of the tasks, and thus, the sizes of the time windows of the partitions. Since these interference delays may lead to deadline violations, they must be bounded during timing analysis.

However, the sharing level implied by multi-core environments leads to an explosion of the number of possible situations to consider to find the Worst Case Execution Times (WCET) of the tasks, so that static timing analysis at code level on multi-core is quite difficult and not solved today. On the other hand, the electronic market is evolving so fast that single-core processors will soon become obsolete, and will not be produced anymore. Avionic system designers will then have no choice but to move to multi/many-core designs. This represents a major challenge for IMA systems, since no consolidated approach to guide the transition of IMA applications from single- to multi-core platforms has been put forward so far.

C. Contributions

In this paper, we propose a framework for automated integration and timing analysis, of legacy IMA software on multi-core processors. To do so, we first present a sufficient schedulability test that is interference-aware, and compatible both with IMA applications and heterogeneous multi-core processors. The test is based on an extension of the response time analysis presented in [12], and computes safe bounds on inter-task memory interference.

As a second contribution, we exploit the resulting IMA, multi-core interference-aware response time analysis, in an optimization framework to automate the SW/HW allocation and scheduling analysis during the system integration phase. The schedulability test is used as a guide to perform design space exploration, and to find an optimized partition-to-core mapping. The optimization criteria we rely on is the minimization of the total workload of the system. By exploiting such an interference-aware schedulability test inside the allocation search, the solution is guaranteed to preserve the feasibility of the final system, despite the additional delays that might appear due to resource sharing.

Finally, our framework enables to produce bounds for the tasks Worst-Case Response Times (WCRT), as well as for the inter-task memory interference. Such bounds are crucial to formally prove the feasibility of the system, but also to determine the minimum requirements for the time window of each partition, which is mandatory in order to set in advance a valid scheduling plan of the global system.

D. *Paper Outline*

The paper is organized as follows: section I introduces IMA systems, our problem statement and contributions. Section II gives some insight on the integration process and avionic requirements. Section III gives an overview of the state of the art related to our work. Section IV presents our system model. Section V then presents our extension of the interference-aware response time analysis to IMA and multi-core contexts. Section VI presents the allocation process implemented in our framework. Section VII presents the implementation and results of our work, and section VIII finally concludes our paper.

II. Avionics Systems

A. *IMA Systems Integration*

During the integration phase of a system design, the software platform is allocated to the hardware platform. The person in charge of this task is called the system integrator. Software functions are usually designed by one or several different suppliers, and are delivered to the system integrator before the beginning of the integration phase. As tasks are scheduled dynamically, their schedule is not set in advance, but each supplier must have verified and validated the schedulability of the tasks of each delivered partition. The job of the system integrator is then to take care of the global schedule, by computing the size and activation offset of the time windows of each partition. We consider legacy code, which means the schedulability analysis performed by each supplier has been verified for single-core environments. Thanks to the sufficient test we present in this paper, the tasks schedulability is verified again during the integration phase, to take into account the fact that partitions will be running in parallel, by including additional inter-task and inter-partition interference delays due to sharing the main memory. One schedulability test that may be used is the response time analysis, which leads to the computation of a bound on the WCRT of each task.

After the production of WCRT bounds, the integrator is able to assess the minimum size required for the time window of each partition: the window must be large enough for the tasks inside the considered partition to complete their execution, even in the worst-case. Once all sizes have been set, the integrator decides of activation dates for the beginning of each time window, starting from the instant when the system is switched on. IMA partitions are periodic, i.e. a time window must be reserved for each partition at a periodic rate. We consider this information as implying not only a period, but also an implicit deadline per partition, equal to the period, since each partition must be assigned a window before its next periodic activation. As a consequence, one important step of the integration process is to verify that, for each partition, time window sizes and activation dates match the periodicity of the partition. If it is not the case, the integrator must consider another SW/HW allocation, for instance by increasing the number of embedded processors for instance. The integrator is likely to have to try different SW/HW allocations before finding a configuration that complies to the requirements of all the described steps.

Eventually, the chronograph resulting from the global schedule consists in the pattern that will be repeated cyclically on the system, for as long as it is switched on. This pattern is also called the major frame. In this paper, we do not address the schedule generation activity of the integration phase.

B. *Avionics Requirements*

Final validation of avionic systems is done through a thorough certification process, shepherded by various regulations and recommended practices (like [5], [4] for instance). In 2014, the Certification Authorities Software Team (CAST) submitted a first position paper regarding multi-core processors [1]; static, assymetric scheduling is recommended, as it enables to reuse existing code without modification, and since global scheduling would require further proofs and analyses when looking for certification. CAST encourages the privatization of shared resources as much as possible. This goes in the sense of IMA needs, where the more private to each partition are the resources, the better it is. As a consequence, we assume the main memory and shared caches are partitioned into areas private to each core. This reduces the number of inter-core interference to consider: dividing shared cache levels into areas private to each core suppresses inter-core cache interference, as tasks on different cores are unable to evict each other's data. However, partitioning the main memory into core-private areas does not suppress inter-core interference, as long as the memory controller is still shared by the cores. Because all requests cannot be handled simultaneously by the controller, it results in waiting delays at runtime, and thus, inter-core interference, even in the case of a main memory privatized at core level. In this paper, whenever we mention main memory interference, we actually refer to the inter-core interference due to sharing the memory controller.

To be in line with CAST multi-core study: (i) we consider static partitioned scheduling for partitions: each partition is statically assigned to a core, where it will be activated from the beginning till the end of the life cycle of the system; (ii) main memory and shared caches are partitioned at core level. However, some avionic functions might implement inter-partition communications. In such cases, we assume memory banks are exceptionally shared, between the two partitions concerned; these banks store the shared data.

III. State of the Art

In the literature, no consolidated approach to guide the transition of IMA applications from single-core to multi-core platforms has been put forward so far, so the problem is still open. To our knowledge, our work is the first to combine, inside one approach for the allocation search: (i) an interference-aware multi-core schedulability test; (ii) the computation of a maximum bound for memory interference per task; (iii) the computation of a maximum bound for the WCRTs of the tasks taking interference into account. In addition, our approach is also the first to consider, at the same time: (i) IMA architectures: tasks are defined inside partitions, which implies additional activities and verifications to be handled during software integration, like the computation of the sizes of the time windows of the partitions; (ii) the possible heterogeneity of the platform: in heterogeneous multi-cores, the execution duration of a piece of code depends on the core it is executed on: as a consequence, each task has one execution time in isolation per core.

Table I summarizes the characteristics of related work. The main characteristic of our contribution is the joint management of the allocation and scheduling analysis, with interference considerations. To our knowledge, only [15] and [10] handle these two activities at the same time. However, the interference-aware schedule generated in [15] relies only on a selection of measured execution times of the tasks when running simultaneously with each other, thus no safe bound on interference per task is produced. In [10], model checking is performed to obtain a reliable schedule, and thus safe bounds on tasks worst-case interference. However the author makes strong assumptions on the hardware architecture, which are currently verified only in the Kalray MPPA multi-core COTS [2], and neither the heterogeneity nor the IMA environment are taken into account. In [9] the interference-aware schedule is produced after data path analysis thanks to a detailed hardware model of the platform, which implicitly leads to interference bounds. The work presented in [13] relies on runtime monitoring to dynamically adapt a scheduling plan, thanks to regular comparisons of the remaining time available for each task before its deadline, and its theoretical remaining execution time required to finish, computed in isolation at observation points. Thus no bound on task interference is given, and the correctness of the final schedule is more difficult to prove. In [6], the authors propose an interference-aware multi-core response time analysis that takes into account inter-task interference due to sharing access to the DRAM, cache and bus resources. The framework does not implement any automatic allocation search process though, and is applicable neither to heterogeneous platforms, nor to IMA software architectures. Eventually, the IMA architecture is only considered in scheduling analyses, and except for the work in [9], the heterogeneity is considered only for the allocation search without interference issues.

IV. Model of the System

Let N_p and N_t respectively denote the total number of partitions and tasks of the software platform, and N_c the number of cores of the multi-core processor of the hardware platform. Tasks are denoted as τ_i, and are ordered by their unique priority levels i: τ_i has a higher priority than τ_j if i is smaller than j. We model tasks as a vector $\tau_i = (C_i, D_i, T_i, H_i)$ with $C_i = (C_i^1 ... C_i^{N_c})$ and $H_i = (H_i^1 ... H_i^{N_c})$. In the vector C_i, each C_i^k is the worst-case execution duration of τ_i when running in isolation on the core k. The C_i^k parameters can be deduced from static WCET analysis of the code, with tools like OTAWA [7] for example. Similarly, H_i gives the maximum number of data requests to the memory that τ_i can issue depending on its core assignment; a bound on each element of H_i can be extracted after static code analysis. T_i is the period of τ_i, and D_i its deadline.

We model partitions as a vector $\pi_i = (E_i, P_i)$, where E_i is the size of the time window to be reserved for π_i, and P_i is the period of π_i. As explained in the introduction, we compute E_i such that the task with the lowest priority level only executes once per window. To ensure an accurate time window size, E_i is computed after the response time analysis, to guarantee a window large enough for the tasks to complete their execution in the worst-case:

$$\forall i, E_i = \max_{\tau_j \in part(i)} (R_j) \tag{1}$$

where $part(i)$ contains the tasks belonging to the partition π_i. Indeed, by definition, the WCRTs correspond to the situation where every task is preempted as many times as possible during its execution. In our case, we consider all tasks of a partition to be released simultaneously at the beginning of each time window. Priority levels are unique, i.e. two different tasks cannot have the same priority level. As such, for a given partition, the task with the lowest priority level has been preempted as many times as possible by all other tasks of the same partition. As a consequence, if τ_j is the task with the lowest priority level of π_i, then R_j accounts not only for the execution time of τ_j, but also for the execution times of all other tasks of the

Related Work	Avionic Systems	Heterogeneity	Allocation	Interf.-Aware Schedule	Interference Bounding
Bradford et al. [9]	yes	yes	no	yes	yes
Paolieri et al. [15]	no	no	yes	yes	no
Baruah et al. [8]	no	yes	yes	no	no
Tamas et al. [16]	no	yes	yes	no	no
Giannopoulou et al. [10]	no	no	yes	yes	yes
Kritikakou et al. [13]	no	no	no	yes	no
Altmeyer et al. [6]	no	no	no	yes	yes
our work	yes	yes	yes	yes	yes

TABLE I: Summary of related work contributions

partition. τ_j is then the task with the biggest WCRT, and can be used to compute the required size of the time window of its partition as described in equation (1).

The task-to-partition mapping is defined by the matrix $PART$ as follows:

$$PART_{ji} = \begin{cases} 1 & \text{if } \tau_i \in \pi_j, \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Inter-partition communications are modeled by the matrix M as follows:

$$M_{ij} = \begin{cases} 1 & \text{if } \pi_i \text{ and } \pi_j \text{ exchange data}, \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Except for E_i, all these parameters are given as inputs to our framework. Additional parameters are computed though. To perform the scheduling analysis, the following variables are defined, for each task τ_i:

- B_i is a maximum bound on inter-task interference due to sharing the main memory;
- R_i is the WCRT of the task;
- $iso(i)$ is the element C_i^k of the vector C_i where k corresponds to the index of the core to which τ_i has been allocated.

These three sets of variables are computed automatically by our framework during schedulability analysis, as will be explained later in this paper.

To perform the allocation search, we implement in our framework the matrix a, defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } \pi_j \text{ is allocated to core } i, \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Eventually, to ease the explanations of the interference mathematical model, we also introduce the following sets:

- $part(i)$ contains the tasks belonging to the partition π_i: $part(i) = \{\tau_j, PART_{ij} = 1\}$
- $core(p)$ contains the tasks belonging to the core p: $core(p) = \{\tau_i, a_{pj} = 1, \tau_i \in part(j)\}$

V. IMA Response Time Analysis

A. Definition

The response time analysis first computes the WCRTs R_i of the tasks τ_i, and then, compares the results with the corresponding deadlines: tasks are schedulable if and only if R_i is smaller than D_i. In non-IMA, single-core systems, R_i is computed as the fixed-point solution of the following iterative equation, defined by M. Joseph and P. Pandya [11]:

$$R_i^{k+1} = C_i^{'} + \sum_{\substack{\forall j, \\ \tau_j \in hp(\tau_i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j^{'} \quad (5)$$

where k is the iteration number and $C_i^{'}$ is the execution duration of τ_i in isolation. For the first iteration, R_i^0 is set to $C_i^{'}$. The second term in (5) refers to the maximum waiting delay due to the preemption of τ_i by tasks with a higher priority. The set containing such tasks is denoted $hp(\tau_i)$.

B. Transposition to IMA, Multi-Core Environments

1) Multi-Core: In equation (5), all tasks of the system are on the same single-core processor. In multi-core environments, the tasks are rather dispatched among the cores of the processor. In theory if interferences are ignored, the situation is equivalent to considering N_c independent single-core processors, so that equation (5) can be easily reused: the set $hp(\tau_i)$ will then refer to the tasks of higher priorities that are mapped on the same core than τ_i, since only the tasks on the same core as τ_i could be able to preempt it. As a consequence, if τ_i is mapped to core p:

$$R_i^{k+1} = iso_i + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ and\, \tau_j \in core(p)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil iso_j \quad (6)$$

with iso_i being equal to C_i^p by definition. To compute iso_i during the WCRT analysis, the core on which τ_i is allocated must be retrieved. However, in IMA architectures, tasks belong to partitions, and it is the partitions – not the tasks – that are assigned to a core. Thus, to compute $iso(i)$: (i) the partition to which τ_i belongs is identified first, (ii) the core to which the corresponding partition has been assigned is identified, and then (iii) $iso(i)$ is deduced as equal to the element of the vector C_i corresponding to the core identified in the second step. The mathematical equation corresponding to the computation of $iso(i)$ is the following:

$$\forall i,\ iso(i) = \sum_{p=1}^{N_p} PART_{pi} \times \left(\sum_{k=1}^{N_c} a_{kp}.C_i^k \right) \quad (7)$$

Indeed, $PART_{pi}$ will be non null only if π_p is the partition to which τ_i belongs; then, the term a_{kp} will be non null only if π_p is allocated to core k, which leads to the identification of the index k of the core on which π_p, and thus τ_i, is mapped, and then to the duration C_i^k.

2) IMA partitions: In IMA architectures, further refinement of equation (6) can be given, according to the two-level schedule. A task can be executed only within the time window of its partition. This implies that the only tasks susceptible to preempt τ_i must belong to the same partition as τ_i. As a consequence, equation (6) becomes:

$$R_i^{k+1} = iso(i) + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ and\, \tau_j \in part(i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil iso(j) \qquad (8)$$

C. Main Memory Interference Model

As mentioned in section I-B, the response time analysis (5) is often compositionally augmented to consider additional latencies likely to appear in practice. In our case, we add to the response time analysis in equation (6), a bound B_i of the maximum interference delay each task can suffer due to sharing the main memory:

$$R_i^{k+1} = iso(i) + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ and\, \tau_j \in part(i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil iso(j) + B_i \qquad (9)$$

As shown in figure 1, the memory is divided into banks, and each bank is divided into columns and rows. When a request is treated by the memory controller, the bank to access is identified first, then the row. When a task issues a memory request, the waiting delay suffered before the request is satisfied depends on the order in which the pending requests are treated by the memory controller. We implemented the DRAM memory model presented in [12], where a detailed model of intra- and inter-bank access delays is given, based on a realistic memory model implementing FR-FCFS (First Ready-First Come First Serve) protocol.

The approach in [12] presents two computation methods, and chooses the minimum value of the two produced bounds for each task. The maximum bounds produced thanks to the two methods will be referred to as $B_{i,method_1}$ and $B_{i,method_2}$ respectively.

We present here the mathematical model of memory interference of [12], as well as our modifications leading to our final response time analysis formulation. Due to lack of space, we will only give a brief description of the model, and for a more detailed explanation, interested readers are invited to read [12].

1) Method 1: Request Driven Approach: In the request driven approach, the maximum delay suffered by a request on one core is assessed. To do so, two types of interference are considered: inter- and intra-bank interference.

For a request req issued by one core p, the worst case situation happens when: (i) every other core issued a request just before req; (ii) none of these requests targets the same memory bank as req; (iii) the treatment of each of these requests take the longest latency possible, l_{max} – which happens when neither the previous bank, nor the previous row were accessed just before, and the type of the request is always different than the preceding one. The computation of l_{max} is explained in detail in [12], and depends on standard DRAM timing parameters that can be found in the DRAM datasheet (see table 1 of [12] for instance). As a consequence, the inter-bank interference delay for a request issued by the core p is:

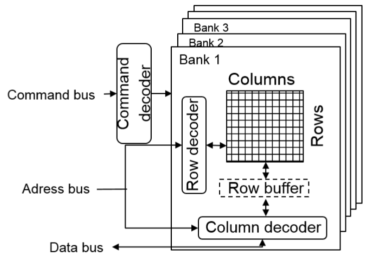


Fig. 1: Model of Memory Banks

$$RD_p^{inter} = \sum_{\substack{\forall q, q \neq p\, and \\ shared(p,q) = \varnothing}} l_{max} \qquad (10)$$

where $shared(p, q)$ is the set of memory banks shared by the cores p and q. In our model, the sets $shared(p, q)$ can be deduced from matrices a and M. In particular, $shared(p, q)$ is empty only if no partition on core q is sharing a memory area with any partition on core p. This translates into the terms a_{pi}, a_{qj} and M_{ij} being equal to zero for all partition π_i belonging to p and all partition π_j belonging to q. As a consequence, the emptiness of $shared(p, q)$ can be assessed in our model as follows:

$$(shared(p, q) = \varnothing) \equiv \left(\sum_{i=1}^{N_p} \sum_{j=1}^{N_p} a_{pi} \times a_{qj} \times M_{ij} = 0 \right) \qquad (11)$$

Equation (10) is the inter-bank interference delays, in the case where no request to the same bank as req was produced. In the case where requests to the same bank as req are issued, the longest interference delay for req to be serviced happens when: (i) all the other cores q that share access to the same bank as core p emitted a request before req; (ii) all these requests concern access to a different row; (iii) a memory reordering is happening. If L is the row-conflict service time, to open a row before accessing a column, then the worst-case delay per such request is $L + RD_q^{inter}$. The intra-bank interference delay suffered by req issued by the core p is thus:

$$RD_p^{intra} = reorder(p) + \sum_{\substack{\forall q, q \neq p\, and \\ shared(p,q) \neq \varnothing}} (L + RD_q^{inter}) \qquad (12)$$

As for l_{max}, L depends on standard DRAM parameters that can be found in the memory's datasheet (see [12] for a detailed description). $Reorder(p)$ computes the delay of req due to the reordering effect (see [12] for a detailed description). Following the same reasoning than in the previous paragraphs, $shared(p,q)$ is non empty only if there exists a partition on core q that is sharing a memory area with a partition on core p. This translates into the terms a_{pi}, a_{qj} and M_{ij} being both equal to one for at least one partition π_i belonging to p and one partition π_j belonging to q. As a consequence, the emptiness of $shared(p,q)$ can be assessed as follows:

$$(shared(p,q) \neq \varnothing) \equiv \left(\sum_{i=1}^{N_p} \sum_{j=1}^{N_p} a_{pi} \times a_{qj} \times M_{ij} \neq 0 \right) \tag{13}$$

Finally, the total maximum interference delay a request originating from the core p can experience is equal to:

$$RD_p = RD_p^{inter} + RD_p^{intra} \tag{14}$$

Each task τ_i of core p having H_i^p requests to issue, the total maximum interference delay directly caused by the issuing of these requests is bound by $H_i^p \times RD_p$. Tasks being preemptible though, the cost of memory requests of tasks with higher priorities has also to be accounted for. Therefore, the bound on τ_i's memory interference is defined as follows:

$$B_{i,method_1} = H_i^p \times RD_p + \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ \tau_j \in part(i)}} \left\lceil \frac{R_i^k}{T_j} \right\rceil H_j^p \times RD_p \tag{15}$$

2) Method 2: Job Driven Approach: In the second method, the author of [12] focuses on how many interfering memory requests are generated during the execution of a task. The maximum number of requests generated by a core p during a time interval t, $A_p(t)$, depends on the number of generated requests by its tasks that are executed during that same time interval:

$$A_p(t) = \sum_{\forall \tau_i \in core(p)} \left\lceil \frac{t}{T_i} \right\rceil H_i^p \tag{16}$$

As an IMA environment, during the execution of τ_i, only the tasks of the same partition as τ_i are eligible to emit requests for a given core. Thus the maximum number of requests that core p can generate during the execution of τ_i is:

$$A_p(t_i) = \sum_{\forall \tau_j \in part(i)} \left\lceil \frac{t_i}{T_j} \right\rceil H_j^p \tag{17}$$

where t_i is the time interval during which τ_i is executed.

Once the maximum number of memory requests generated during a given time interval has been expressed thanks to the definition of A_p, one can express inter- and intra-bank interference delay imposed on a core p during a time interval t, respectively:

$$JD_p^{inter}(t) = \sum_{\substack{\forall q, q \neq p \; and \\ shared(p,q) = \varnothing}} A_q(t) \times l_{max} \tag{18}$$

$$JD_p^{intra}(t) = \sum_{\substack{\forall q, q \neq p \; and \\ shared(p,q) \neq \varnothing}} \left(A_q(t) \times L + JD_q^{inter} \right) \tag{19}$$

Finally, to consider the maximum memory interference delay τ_i can suffer, equations (18) and (19) compute the elapsed time between the beginning and the end of the execution of τ_i. To produce a bound that is safe in the worst-case situation, the time interval to consider is R_i. As a consequence, if τ_i is allocated on the core p, then the maximum bound for the memory interference it can experience during its execution is given by:

$$B_{i,method_2} = JD_p^{inter}(R_i) + JD_p^{intra}(R_i) \tag{20}$$

D. Final Definition of R_i and B_i

Methods 1 and 2 being about the computation of maximum bounds, B_i is set to the less pessimistic of the two:

$$B_i = \min \left(B_{i,method_1}, B_{i,method_2} \right) \tag{21}$$

Eventually, the final formula to compute safe bounds on the WCRTs to perform the analysis is given in equation (9), where B_i is computed according to equation (21).

VI. Allocation Process

During the allocation process, the system integrator decides on which processor each partition will be executed. In the case of assymetric scheduling on a multi-core processor, the integrator should decide on which core each partition will be allocated. An allocation then refers to a partition-to-core mapping, each partition being supposed to run on the same core from the beginning till the end of life of the system.

After an allocation has been chosen, the system integrator then sets a global schedule for each core of the multi-core. To do so, the integrator first computes the size of each partition time window, and then sets an offset for the activation of each window inside the major frame of each core. The size of a time window depends on the tasks supposed to run within its boundaries: the window should be large enough for all these tasks to complete their executions. The offset of a window should be chosen so that the end of the window is before the beginning of the next period of the partition.

If these conditions are verified, the resulting global schedule on each core are considered to be valid, as well as the partition-to-core allocation that led to such schedules. If on the contrary, it is not possible to find a valid schedule for at least one core of the multi-core, another allocation must be selected. The allocation phase is thus important, since the validity of the computed global schedules depends on it.

We propose to automate the SW/HW allocation process of the design phase of a system. To do so, we perform an exhaustive search for a valid allocation using constraint programming. The constraints represent the scheduling requirements of the system. To be able to do so, we rely on equation (9) to guide the search. Figure 2 summarizes our exploration process:

a) Step 1: the first step consists in choosing an allocation, in order to evaluate if it is a potential solution for the SW/HW integration. The allocation is chosen automatically by the framework, among all the possible combinations. Eventually, all allocations will be evaluated during the exploration process.

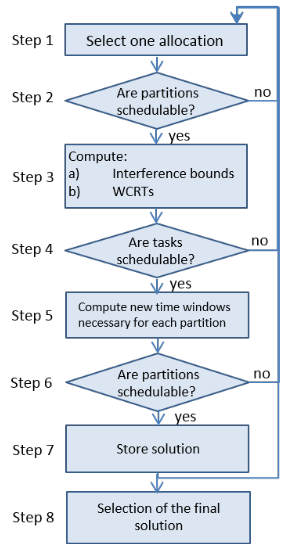


Fig. 2: exploration process

b) Step 2: The second step represents an early verification that each partition window can end before the next period. This cannot be verified with certainty before knowing the activation offsets in the global schedule, nor before computing the tasks WCRT with equation (9). However, if one window happens to be larger than the period of the corresponding partition, then the selected allocation could never lead to a valid global schedule. As a consequence, we add the following constraints, to help the early rejection of such invalid allocations:

$$\forall i, E_i \leqslant P_i \tag{22}$$

where E_i is computed according to equation (1), and each R_i with equation (8). If this condition is not respected, the search process goes back to step 1 and a new allocation is chosen.

c) Step 3: If step 2 is successful, R_i and B_i are computed for all tasks thanks to equations (9) and (21).

d) Step 4: The schedulability of the tasks in the multi-core allocation currently under evaluation is assessed. Tasks are schedulable if and only if:

$$\forall i,\ R_i \leqslant D_i \tag{23}$$

e) Step 5: If step 4 is successful, we update the sizes E_i of the time windows of the partitions: now that the interference-aware WCRTs of the tasks are available, we have to check that the time windows are still large enough in the worst-case, i.e. when the durations of the tasks are equal to their R_i. If it is not the case, the window size of the corresponding partitions are increased accordingly thanks to equation (1).

f) Step 6: The sixth step is based on the same principle as the second step, evaluated with the new E_i values. If a window E_i is larger than the corresponding period P_i, it means that the currently assessed allocation may lead to deadline violations due to memory interference. The search process then rejects the currently evaluated allocation as invalid, and goes back to the first step to choose a new allocation.

g) Step 7: If the sixth step is successful though, the current allocation is stored as a valid solution.

h) Step 8: The last step of the process is the selection of a solution among the valid allocations that were stored at step 7. We propose to do so according to an optimization criteria. Our framework selects the solution that minimizes the total workload of the system. We justify our choice by the fact that this parameter is a performance characteristic, but also because it is proportional to inter-task interference delays, which add up to the tasks executions. Because of these delays, (i) the extra time available before tasks deadlines are reached at runtime is drastically shortened, and (ii) the integrator is given less flexibility for the setting of a valid global schedule for each core in the considered allocation. The objective function we defined in our framework is the following:

$$minimize \sum_{i=1}^{N_t} \frac{R_i}{T_i} \tag{24}$$

VII. Implementation

We implemented the optimization problem proposed in this paper, to assess the benefits that can be drawn from its use. To do so, we compared several qualities of the solution returned by our approach, to the solutions that would have been returned by ”classic” approaches, meaning approaches without interference consideration. As explained before, no such classic approach exists, so we modified a copy of our optimization problem into a classic allocation problem: we remove the interference bound in equation (9), but we still compute it thanks to equation (21), to get the corresponding interference. The resulting process is referred to as ”classic approach” or ”interference-oblivious solving” in the rest of the paper. We implemented the optimization problem in CPLEX [3], that has been running on a computer with an Intel Core i7 2.20 GHz processor with 16Gb of RAM.

A. Presentation of the Case Study

We derived a case study from the report [14], which presents a specification of an avionics Mission Control Computer (MCC) system. No mention is made about an IMA architecture, nor about maximum numbers of requests to the main memory by the tasks, but for the sake of the analysis, (i) we consider each function to correspond to a partition, as if each one had been designed by a different supplier; (ii) we randomly generate H_i^k values for each task, ranging from low to intensive usage of the main memory, respectively 1 and 40 requests per microsecond [12]. We also built the C_i vectors by deriving the execution time in isolation given for each task in the report. The data corresponding to our case study is summarized in table II.

Partition	tasks C_i (ms)	T_i (ms)	D_i (ms)	P_i (ms)
1	C_1=(8, 7.2, 7.6, 6.4)	55	55	480
	C_2=(6, 5.4, 5.7, 4.8)	80	80	
2	C_3=(2, 1.8, 1.9, 1.6)	40	40	480
	C_4=(2, 1.8, 1.9, 1.6)	80	80	
	C_5=(2, 1.8, 1.9, 1.6)	480	200	
3	C_6=(4, 3.6, 3.8, 3.2)	40	40	480
	C_7=(1, 0.9, 0.95, 0.8)	480	40	
	C_8=(2, 1.8, 1.9, 1.6)	480	40	
	C_9=(1, 0.9, 0.95, 0.8)	480	200	
4	C_{10}=(1, 0.9, 0.95, 0.8)	10	5	480
	C_{11}=(7, 6.3, 6.65, 5.6)	100	100	
	C_{12}=(1, 0.9, 0.95, 0.8)	480	200	
	C_{13}=(1, 0.9, 0.95, 0.8)	4800	200	
	C_{14}=(2, 1.8, 1.9, 1.6)	480	400	
	C_{15}=(6, 5.4, 5.7, 4.8)	480	400	
5	C_{16}=(1, 0.9, 0.95, 0.8)	1920	40	1920
	C_{17}=(1, 0.9, 0.95, 0.8)	1920	40	
	C_{18}=(6, 5.4, 5.7, 4.8)	52	52	
	C_{19}=(6, 5.4, 5.7, 4.8)	52	52	
	C_{20}=(8, 7.2, 7.6, 6.4)	52	52	
	C_{21}=(1, 0.9, 0.95, 0.8)	100	200	
	C_{22}=(2, 1.8, 1.9, 1.6)	1000	1000	
	C_{23}=(1, 0.9, 0.95, 0.8)	1920	200	
	C_{24}=(1, 0.9, 0.95, 0.8)	1920	200	
6	C_{25}=(1, 0.9, 0.95, 0.8)	100	200	480
	C_{26}=(2, 1.8, 1.9, 1.6)	480	400	
7	C_{27}=(2, 1.8, 1.9, 1.6)	200	200	480
	C_{28}=(3, 2.7, 2.85, 2.4)	480	100	
8	C_{29}=(5, 4.5, 4.75, 4)	1000	400	1920
	C_{30}=(1, 0.9, 0.95, 0.8)	1920	200	
	C_{31}=(10, 9, 9.5, 8)	1920	800	

(a) Description of the tasks and partitions

H_i used for the first test
H_1 = (160000, 144000, 152000, 128000)
H_2 = (30000, 27000, 28500, 24000)
H_3 = (42000, 37800, 39900, 33600)
H_4 = (70000, 63000, 66500, 56000)
H_5 = (44000, 39600, 41800, 35200)
H_6 = (80000, 72000, 76000, 64000)
H_7 = (21000, 18900, 19950, 16800)
H_8 = (14000, 12600, 13300, 11200)
H_9 = (15000, 13500, 14250, 12000)
H_{10} = (12000, 10800, 11400, 9600)
H_{11} = (133000, 119700, 126350, 106400)
H_{12} = (2000, 1800, 1900, 1600)
H_{13} = (26000, 23400, 24700, 20800)
H_{14} = (20000, 18000, 19000, 16000)
H_{15} = (6000, 5400, 5700, 4800)
H_{16} = (1000, 900, 950, 800)
H_{17} = (26000, 23400, 24700, 20800)
H_{18} = (192000, 172800, 182400, 153600)
H_{19} = (240000, 216000, 228000, 192000)
H_{20} = (80000, 72000, 76000, 64000)
H_{21} = (7000, 6300, 6650, 5600)
H_{22} = (52000, 46800, 49400, 41600)
H_{23} = (34000, 30600, 32300, 27200)
H_{24} = (29000, 26100, 27550, 23200)
H_{25} = (14000, 12600, 13300, 11200)
H_{26} = (24000, 21600, 22800, 19200)
H_{27} = (48000, 43200, 45600, 38400)
H_{28} = (60000, 54000, 57000, 48000)
H_{29} = (135000, 121500, 128250, 108000)
H_{30} = (340000, 306000, 323000, 272000)
H_{31} = (10000, 9000, 9500, 8000)

(b) Vectors H_i used in the first test

TABLE II: Case Study Data

B. Tests Performed

As we will explain in detail in the next paragraphs, we performed two different experimentations. In both cases, our goal is to compare the solution returned by our framework with the solution that would have been provided by interference-oblivious processes. The comparison is made on the basis of the workload reduction achieved thanks to our framework, but also on the interference percentage, and the slowdown suffered by the tasks, in the selected solution.

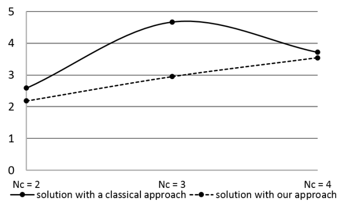
1) Test 1: In the first test, the number of cores N_c of the multi-core platform is the only variable parameter. We ask, both our framework and a classic allocation search, to find an allocation of the eight partitions described in table II on N_c cores. The H_i vectors used in the first test are described in table IIb. The maximum possible number of cores is eight, corresponding to the situation where each partition is allocated alone on a core. The result of the allocation search is either a valid SW/HW allocation, or the answer that there exists no valid solution for the number of cores considered. As we will explain in detail in subsection VII-C "Results", no solution was found for N_c greater than or equal to five; that is the reason why the C_i and H_i vectors in table II only contain four elements each, and not eight.

2) Test 2: Since memory interference depends on the H_i values, the comparison results and the realized performance gains are very data-dependent. To get a general appreciation of the optimization gain achievable thanks to our framework, we test different intensities of memory utilization by the tasks. Such a second test also enables to see the evolution of inter-partition interference with the intensity of the memory usage by the tasks. As mentioned earlier, memory utilization intensity varies approximately between 1 and 40 requests per microsecond per task. As a consequence in this second test, there are two variable parameters: N_c and the vectors H_i. We use data from table IIa but not from table IIb, and we build the vectors H_i as follows instead, for all tasks τ_i and all cores k:

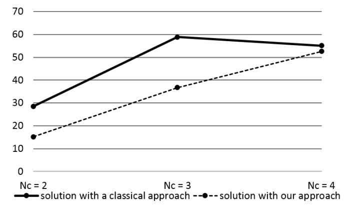
$$H_i^k = C_i^k(in\,\mu s) \times x, \quad x \in \{1, 10, 20, 30, 40\} \quad (25)$$

C. Results

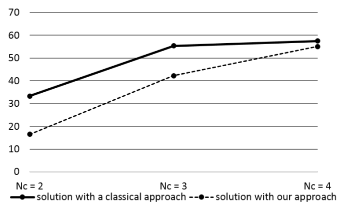
For both tests, our framework was unable to find solutions for N_c greater than 4. On the contrary, in the classic approach, the search always finds a solution, but after analysis of the output memory interference bounds B_i, the solution appears not to be valid in reality: some bounds lead to actual WCRTs being bigger than the deadline of the corresponding tasks, which invalidates the feasibility of the selected solution. The difference between our framework and the classic approach being the interference consideration, we can draw the conclusion that our framework prevents from choosing allocations that, later on during the schedule planning phase, appear to be infeasible. This also implies that our work enables to reduce the time spent during the last phases of a system design.



(a) total workload



(b) interference percentage in the total workload



(c) median slowdown percentage

Fig. 3: Results of the first test

In terms of computation time, solutions were always found in less than 10 seconds with two cores, 62 seconds with three cores and 7 minutes with four cores.

1) Test1: Figure 3 shows the results of the first test. According to the figure, our framework always returns a better solution than the classic approach, the difference between the two solutions being more noticeable for three cores. We were able to show some significant performance enhancement, with up to 36.9% of workload reduction, 46.6% of interference reduction and 50.4% slowdown reduction.

2) Test 2: Figure 7 displays the results of the second test. The graphs on the left show the results corresponding to our framework, whereas the graphs on the right correspond to the solution with classic approaches for the allocation search. The same legend is used in all graphs, to ease results interpretation. In all runs performed, the solution found with our framework is better than with classic approaches, by allowing up to 39.6% workload reduction, 54.5% interference reduction and 44.4% slowdown reduction.

Eventually, for a given value of N_c, one can see the evolution of workload, interference and slowdown depending on the intensity of memory usage by the tasks. All three parameters increase with the usage intensity. The only exception is for four cores in figures 4b and 5b: workload and interference obtained with x equal to 40 in equation (25) are respectively lower than with x equal to 30, and is quite close to the solution found by our framework. Memory interference can only increase with the number of memory requests, since tasks using intensively the main memory are more likely to suffer more from interference than the tasks that only perform a small number of requests to the memory per execution. This implies that the classic approach managed by chance to find an optimized solution for x equal to 40, but failed to do so for x smaller than 40. This can be considered as an expected observation, since the objective function (24) in classic approaches ignores the B_i terms in the computation of the WCRTs.

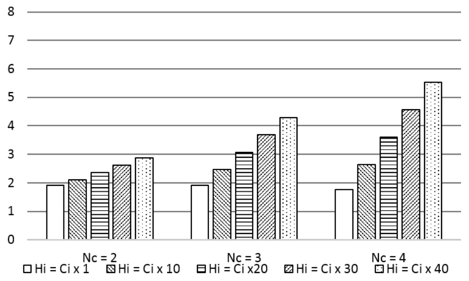
Another remark for a given number N_c can be made when we compare our framework to interference-oblivious search. The speed of the increase of workload, interference and slowdown respectively is slower in figures 4a, 5a and 6a than in figures 4b, 5b and 6b, which is an interesting quality for a system integrator. The difference of speed is more visible in the graphs of figures 4a and 5a than in figure 6a though. This is due to the fact that our optimization criteria is the workload reduction. In the future, it might be interesting to experiment multi-objective cost functions, to have multiple parameters guiding the search and solution selection.

VIII. Conclusions and Future Work

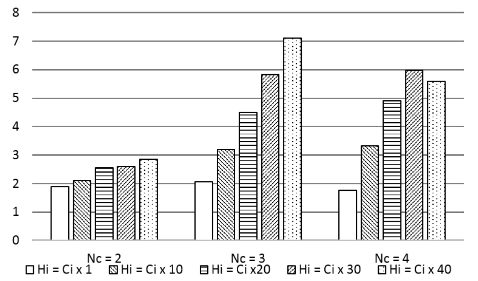
In this paper we have presented a derivation of the response time analysis in [12] to adapt it to IMA systems and heterogeneous multi-core platforms. The resulting response time analysis gives a sufficient schedulability test for IMA applications in heterogeneous multi-core environments. We then proposed to reuse this schedulability test to guide the design space exploration during the SW/HW integration design phase of IMA systems. To do so, we formulated an optimization problem to perform a timing-aware SW/HW allocation search. On the two implemented tests, our results showed that our approach enabled a real performance gain, by achieving up to 54.5% workload reduction, 54.9% memory interference reduction, and 50.4% slowdown reduction at runtime. In the future, we plan on refining the interference model, in order to take more than just main memory interference into account.

References

[1] CAST-32, ”Multi-core Processors”, 2014.
[2] ”http://www.kalrayinc.com/”.
[3] IBM ILOG, Cplex CP Optimizer. Website: http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/.
[4] RTCA Radio Technical Commission for Aeronautics, ”Software Considerations in Airborne Systems and Equipment Certification”, 1992.
[5] SAE International, ”Aerospace Recommended Practice ARP4754 – Guidelines For Development Of Civil Aircraft and Systems”,issued 1996; published 2010.
[6] Sebastian et al. Altmeyer. A generic and compositional framework for multicore response time analysis. In *Real-Time Network and Systems (RTNS), 2015 23th International Conference on*, pages 129–138. ACM, 2015.
[7] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: an open toolbox for adaptive WCET analysis. In *Software Technologies for Embedded and Ubiquitous Systems - 8th IFIP WG 10.2 International Workshop, SEUS 2010, Waidhofen/Ybbs, Austria, October 13-15, 2010. Proceedings*, pages 35–46, 2010.

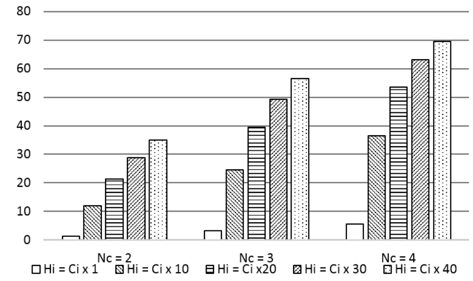


(a) in the solution obtained with our framework

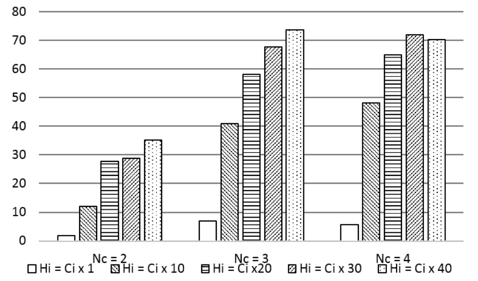


(b) in the solution obtained with classic approaches

Fig. 4: average workload per core (%)

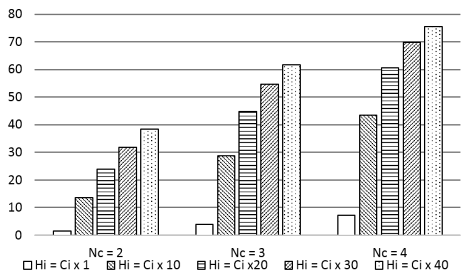


(a) in the solution returned by our framework

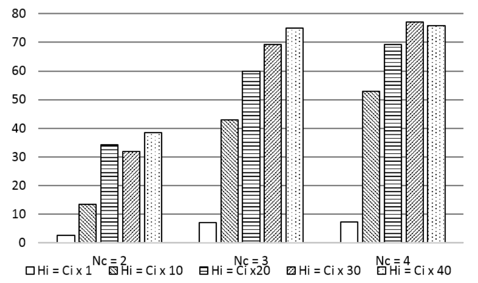


(b) in the solution returned by classic approaches

Fig. 5: average interference due to memory sharing (%)



(a) in the solution returned by our framework



(b) in the solution returned by classic approaches

Fig. 6: average inter-task slowdown (%)

Fig. 7: Results of the second test: our approach (4a; 5a; 6a) versus classic approach (4b; 5b; 6b)

[8] Sanjoy K. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *33rd International Conference on Parallel Processing (ICPP 2004), 15-18 August 2004, Montreal, Quebec, Canada*, pages 467–474, 2004.

[9] Richard Bradford, Shana Fliginger, Rockwell Collins, Cedar Rapids, Sibin Mohan, Rodolfo Pellizzoni, Cheolgi Kim, Marco Caccamo, Lui Sha, et al. Exploring the design space of ima system architectures. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5–E. IEEE, 2010.

[10] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Mapping mixed-criticality applications on multi-core architectures. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6, 2014.

[11] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[12] Hyoseung Kim, Dionisio de Niz, Björn Andersson, Mark H. Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pages 145–154, 2014.

[13] Angeliki Kritikakou, Christine Rochange, Madeleine Faugère, Claire Pagetti, Matthieu Roy, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, page 139, 2014.

[14] Douglas Locke, Lee Lucas, and John Goodenough. Generic avionics software specification. Technical Report CMU/SEI-90-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[15] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Robert I. Davis, and Mateo Valero. IAˆ3: An interference aware allocation algorithm for multicore hard real-time systems. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pages 280–290, 2011.

[16] Domitian Tamas-Selicean and Paul Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*, pages 24–33, 2011.

Session 3

Model Checking

Wednesday 27th, 11:00 – Ariane 1

Formal Specs Verifier ATG: a Tool for Model-based Generation of High Coverage Test Suites

Orlando Ferrante, Marco Marazza, Alberto Ferrari
ALES - UTSCE,
Piazza della Repubblica, 68 - 00185, Roma - Italy
e-mail: name.surname@utsce.utc.com

***Abstract*—In this paper we describe Formal Specs Verifier Automatic Test Generation, a tool generating high coverage test suites for embedded systems. Our tool implements a test case synthesis algorithm using a combination of model checking and optimization techniques starting from a Simulink/Stateflow model of the System Under Test. The main contributions of this paper are the following: we (1) give an extended description of our test generation algorithm, (2) describe the algorithm implementation as part of the Formal Specs Verifier framework, (3) present a concrete application of the tool to a cruise control case study and discuss experimental results comparing our algorithm with a state-of-the art COTS tool.**

***Keywords*—Model-based Automatic Test Generation, High Coverage Test Suites, Formal Methods**

I. Introduction

Testing complex hardware-software embedded systems' architectures is one of the most important and costly phases of their entire development life cycle. A significant portion of development time is spent indeed for the verification and validation phases (V&V) during which teams of test engineers aim at discovering requirements' misinterpretation and implementation errors. Early error detection facilitates correction complexity, which in turn favors minimizing the overall system development cost and time. Testing consists in the execution of a set of predefined input vectors on the System Under Test (SUT) and observation of its response for detecting possible deviations from the expected behavior. In order to perform the testing phase, test vectors (or test cases) must be provided to the test execution environment. A test vector consists of a sequence of pairs (*inputs*, *outputs*) where *inputs* is the set of values to be applied to the system and *outputs* is the set of expected output values. A set of test cases is said to be a *test suite*. In case the execution of the system produces a set of output values not matching the expected ones the test can be used to highlight an error of the SUT. To quantitatively evaluate the quality of test suites, hence of the entire test execution, coverage metrics are used. Coverage metrics measure the effectiveness of a test suite i.e. how much it covers: (1) the structure of the SUT, as in MC/DC coverage metrics, (2) the set of requirements, as in requirements-based testing, or (3) a meaningful subset of admissible faults of the system, as in fault-injection testing. A good test suite satisfies as much as possible a given coverage metric. As a general rule the more the tests executed, the higher the confidence about the correctness of the system, as long as the executed test cases are of high quality. However, the generation of high quality test suites has a relevant impact on costs, e.g. complexity, setup time and execution time; hence, besides providing test suites maximizing a given coverage metric (optimality), it is of paramount importance to ensure a reduced number of test cases (efficiency). Typically, a test suite represents a trade-off between optimality and efficiency. Several techniques have been adopted for the test generation process, especially in the area of test generation from system models whose main idea is to represent the SUT using a formal model and use automatic algorithms for the definition of test cases (model-based test generation). In particular, the use of model checking techniques for test generation has been extensively explored: the test generation procedure is formulated as a problem of reachability, enriching the model with test objectives. Such test objectives must be achieved by the produced test suite in order for a given coverage metric to be satisfied. Once the model is enriched with test objectives, a model checker is used to analyze their reachability. In case a test objective is reachable, a counter-example is produced and stored as a test case, which is said to *cover* that test objective. Once all test objectives are covered, the generated set of tests is a high coverage test suite. In this paper we describe a novel approach for the automatic test generation from system models combining bounded model checking and optimization to generate efficient test cases and test suites. This paper is an extension of the algorithm presented in [1]; the main contributions of current paper can be summarized as follows: we (1) provide an extended description of the test generation algorithm, (2) describe the implementation of the algorithm in our Formal Specs Verifier framework and (3) show the application of our tool to a cruise control case study, along with additional experimental results. The proposed use case only covers the MC/DC as an example. The paper is organized as follows: Section II stresses the central role played by model-based test generation and model-based testing in current industry development processes. Section III summarizes existing approaches for test case generation using exhaustive search techniques. Section IV provides a formal description of the problem our algorithm tries to address and Section VI describes our algorithm in details. Section VII provides an overview of the algorithm implementation as well as an application to a cruise control example and finally Section VIII concludes the paper.

II. Model Based Test Generation

Current practice in industrial systems' design and development process is to create *models* –also known as virtual prototypes– of the system being developed. A model is the artifact meant to imitate the system of interest. The system can be abstracted (modeled) at different levels of refinement and each level of refinement has its own best-fitting formal models. For example, a mathematical model is a set of formal definitions and mathematical formulas that describe the system under analysis. At the various system development phases, models are gradually refined until these get detailed enough to allow for physical implementation of the system. The system implementation can be achieved by means of either fully-automated or pseudo-automated synthesis tools. With regards to testing, the main benefits of model-based approach are manifold: (1) tests can be automatically generated from models (model-based automatic test generation), (2) testing can be performed at different refinement levels, far before the system gets implemented and, more interestingly, (3) tests generated at early phases of the system design can be refined and re-used in later phases of the design to check whether all system properties are still fulfilled (back-to-back testing). Virtual prototyping and related testing are particularly important when the realization of components involves different technology suppliers and manufacturers, and requirements fulfillment must be checked by all the parties at all stages of the design process. This work targets model-based generation of test suites used to bring evidence that functional and non-functional metrics have been achieved by the design implementation.

III. Related work

Several techniques have been described in literature on the topic of automatic test generation. In this section we provide a brief summary of the available techniques pointing out the differences with the one proposed in this paper. Evolutionary and genetic approaches [2] generate tests by randomly exercising the inputs of the SUT and measuring the quality of the test maximizing an objective function. Such function is derived from a structural analysis of the SUT, hence a test that provides better values of the cost function is selected in a set of possible generation process outcomes. In addition, evolutionary testing employs evolutionary algorithm techniques for selection and generation of new tests from a previously generated set of tests. These techniques rely on random search adding structure to the search to avoid generating low coverage tests. However, there are no guarantees that a selected behavior is effectively the best test case with respect to a given coverage metric. Model-checking based techniques are heavily studied for the generation of test cases. In [3] the authors describe the use of a model checking engine for the generation of test cases starting from a modified version of the SUT model enriched with a reset transition (i.e. a variable that, when true, resets the entire state of SUT to the initial state) and test objectives related to the coverage criteria. The proposed approach differs from ours in several ways. At first, the number of satisfied objectives is a non-deterministic result of the execution of the algorithm hence each test case may cover an arbitrary number of test cases, whereas our method allows for selecting the maximum number of satisfiable test objectives at each execution. In addition, the generated tests are produced monolithically starting from the initial model not allowing the application of an incremental test generation methodology; hence, the applicability of the technique to concrete industrial size cases heavily depends on the size of the input model [4]. In [5] a methodology to generate tests from a formalization of requirements based on a tabular representation is described and the counter-example finding capabilities of model checkers is described as a technique to derive such counter-examples. The use of the model checker is straightforward and there is not any guarantee on the quality of generated tests. In [6] the authors describe a method for extracting test cases from Statecharts state machines considering state transitions covering (i.e. covering all states and/or all transitions of the Statecharts). The generation is executed by enriching the model with additional states and searching from counter-examples of a specific CTL formula. The method may produce redundant test cases and after test generation an additional test reduction phase is needed. This approach differs from ours in that it does not guarantee the generation of a minimal set of high coverage test cases and it relies on a specific CTL-based formulation of the problem. As a consequence it cannot be solved using SAT-based model checking techniques, that usually perform better than BDD-based techniques when a counter-example search is performed. In [7] a method exploiting random-simulation and formal verification is described based on a semi-formal method for the traversing of Extended Finite State Machines (EFSMs) that allows for reaching deep test cases exploiting the strength of simulation-based approaches. However, no guarantees are given in terms of efficiency of the generated suite and the use of the model checking engine is limited to the analysis of constraints to guide the main simulation-based test generation algorithm. In [8] the authors propose a test generation method based on the reachability of given test cases as well as extension of already generated test cases. This approach differs from ours in several ways. First, it does not provide guarantees that every generated test is optimal with respect to a specific coverage metric. The model checker is called in order to satisfy some of the test goals but no guarantees are provided to the number of the goals satisfied. Second, it does not employ an incremental test generation procedure: at every execution no guarantees are given about the length of the found counter-example: there might be cases in which a high coverage test case has a given length, but the same results would be obtained with a shorter test case.

IV. Problem Formulation

A. Model And Coverage Criteria Formulation

We focus on the problem of generating high coverage test suites for discrete time models formalized as connections of blocks. Several languages are available to capture embedded control systems using this formalism, e.g. MATLAB Simulink/Stateflow and Esterel SCADE[1]. A model $\mathcal{M}$ can be formally represented as a connection of blocks exposing a well-defined interface in terms of input and output ports, each describing its run-time behavior as an Extended Finite State Machine (EFSM). Block's interfaces are formalized as a pair of input and output vectors $u[k] \in X, y[k] \in Y$, where X and Y represent the vector domains and $k \in \mathbb{N}$ the discrete time. Each block behavior is then formalized as a transition function $F[u, x, f, k]$ that at each discrete time k maps the vectors input $u[k]$ and current state $x[k]$ to an output vector $y[k]$ and next-state values $x'[k]$, where $x'[k] = x[k+1]$. In order to be fully specified, an initial value for the state vector should be provided. Connections between blocks act as constraints between the values of inputs and outputs that must match at every time step. The transition relation of a block

[1]http://www.esterel-technologies.com/

contains logical and arithmetic expressions for Boolean and bit vectors. For a complete description of the richness of the Simulink/Stateflow languages please refer to [9]. A test suite Π is a set of test traces $\pi_1, \ldots, \pi_n$, each representing a finite sequence of admissible input values: $\pi_k = \langle u[1], \ldots, u[m] \rangle$. Each test trace π_k has a finite length $\|\pi_k\| = m$ that corresponds to the maximum time step for which the model is exercised. The objective of test generation is to produce a test suite that exercises the model for maximizing a given coverage criterion. Several coverage criteria have been defined depending on the test generation algorithm input. As an example for finite state machine a relevant coverage criteria is related to the capability of exercising the state machine enabling as much transitions as possible (transition coverage criteria). In model-based approaches several criteria can be defined depending on the input model. In our flow, Simulink and Stateflow models are processed and we use the following coverage criteria (that can be applied to similar languages such as Esterel SCADE). **MC/DC coverage**: following the definition of MC/DC for software a similar criterion has been followed for model elements associated to Boolean formulas. More precisely, for each block corresponding to a Boolean formula $y = f[u_1, \ldots, u_N]$ to fully satisfy the criterion there should exist a trace such that each input affects the truth value of the output independently of the other inputs [10]. **Relations coverage**: for each block that compares two values $y = u_1 \diamond u_2$ (where $\diamond \in \{\leq, <, =, \neq, >, \geq\}$), there should exist tests for which the output value y transitions from FALSE to TRUE and vice-versa. **State coverage**: for state machines, there should be tests such that the state machine states assume all possible values. **Transition coverage**: for each state machine, there should be tests such that all the transitions of the state machines are asserted.
In our flow, Simulink and Stateflow models are processed and we use the following coverage criteria (that can be applied to similar languages such as Esterel SCADE): MC/DC coverage, relations coverage, state coverage and transition coverage. The first two criteria apply also to guards and actions of the state machines; hence, if a guard is a composite Boolean expression, the test suite should provide tests covering the guard with a maximum MC/DC percentage value.

B. Problem Formulation

The test generation problem we address in this paper can be now formalized as follows. Given a discrete time model $\mathcal{M}$, a coverage metric and a test generation maximum time step m, generate a test suite $\Pi = \{\pi_1, \ldots, \pi_n\}$ of vectors of length $\|\pi_k\| \leq m$, such that:

1) the achieved coverage is *maximal* with respect to the coverage metric
2) each test trace contributes to *increase* the coverage of the test suite

Objective 1) ensures that the generated test suite achieves high test effectiveness, provides high coverage of the input model and captures as much errors as possible, whereas objective 2) ensures that the cost of test execution is well-balanced, meaning that each test trace effectively improves the overall coverage of the model and avoids execution of useless tests.

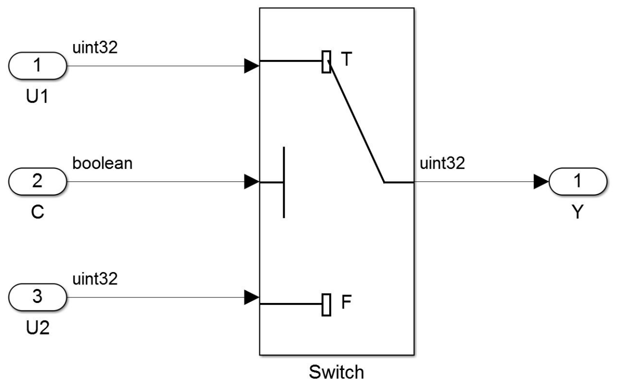


Fig. 1. Test Objective example

V. Maximal Coverage Test Case Generation

VI. Algorithm Description

In this section the test generation algorithm presented in [1] is summarized and extended showing the usage of monitor variables for the automatic synthesis of test cases. In Section VI-A the algorithm is summarized, whereas in Section VI-B the details of the efficient search sub-activity are provided. This section describes the details of the efficient search sub-activity of the test generation algorithm we presented in [1] and differs from a previous implementation [11] generating Minimal Critical Failure Sets.

A. Test Generation Flow

The overall flow of the test generation algorithm is described in Fig. 2. The initial formal model is elaborated in a model transformation step that enriches the input adding a finite number of *test objectives*. Each test objective represents a Boolean condition over the discrete time that should be satisfied by at least one generated test case. The formal description of the test objective depends on the coverage criteria selected by the user (e.g. MC/DC, relation coverage, transition coverage, etc.). During the test objectives generation step the input model is instrumented with additional Boolean expressions representing the test objectives the Automatic Test Generation (ATG) step should satisfy. Each test objective is derived by analyzing the input model blocks according to the selected criteria. As an example consider the Simulink model in Fig. 1 containing a Comparator block and a Switch block. The Comparator has an output that is true when input u_1 is greater or equal to zero and false otherwise, while the Switch connects to the output the first input when the control input c is FALSE and the second input (u_2) otherwise.
In case the user selected the *relation coverage* criteria, two Test Objectives (TOs) to be satisfied would have been derived for the Comparator block: the first TO is true when the comparator output transitions from FALSE to TRUE and the second is true when it transitions from TRUE to FALSE. Similarly, when the MC/DC criterion is selected, two additional test objectives are derived: one that is true when the control input of the Switch transitions from FALSE to TRUE and another one for the opposite condition. Using this approach we are able to convert different coverage criteria to a common criterion (test objective criterion) and the sub-sequent test generation step will try to produce a high coverage test suite maximizing the number

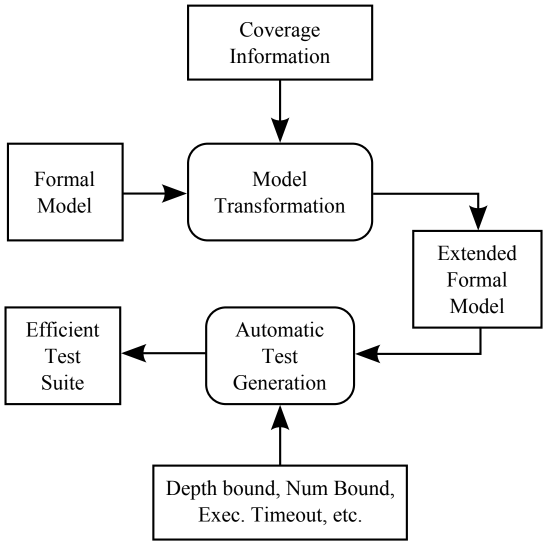


Fig. 2. Test Generation Flow

of test objectives satisfied by the generated traces. Once the extended model is produced, the Automatic Test Generation algorithm generates a set of test traces. Each trace has a finite length and satisfies the maximum number of test objectives that have not been yet satisfied by other (previously generated) test cases. As stated previously, the generated test cases verify the following properties:

- The test suite coverage percentage always increments with the addition of a new test case;
- Each test case is not redundant: no other test case of the same length covering the same (or a super-set of the) set of satisfied test objectives exists.

The ATG process can be bounded in time or in number of generated test cases by setting a set of parameters controlling its execution.

The algorithm is described in Algorithm 1. The algorithm is described in [1] and is summarized here for clarity. At start-up the explored depth bound is set to the initial value (it might be 1 or user-defined). The algorithm loops until all test objectives are satisfied or a given resource/time/depth bound is reached and in each iteration a subset of not yet satisfied test objectives (TO) is identified. The subset may be the entire set of unsatisfied TOs or it may be driven by the user needs (i.e. all the TOs related to the coverage of a given sub-function of the SUT, etc.). Once the TOs are selected an inner loop is performed until an explicit exit condition occurs or all the test objectives selected have been covered. In the inner loop, the formal engine is queried to find all the test cases of length equal to l that maximize the number of satisfied TOs among the ones selected at previous step. Each test case satisfies a unique subset of TOs (i.e. there are not two test cases satisfying the same subset of TOs). If new test cases are found, collect all the subsets of satisfied TOs and remove them from the search to avoid search again their satisfaction in next iteration. In case no new test cases are found we can assess that for the given length bound, test cases satisfying the selected test objectives do not exist. This claim is possible because of the exhaustive search

Algorithm 1 High Level View of the Test Generator Algorithm

```
Input: $\Theta = \{TO_1, \dots, TO_J\}$
Input: $L$ max explored step, $T_{OUT}$ Algorithm execution timeout
    Init
    $l \leftarrow 1$, Currently Explored Length index
    $\mathcal{TC}[0] \leftarrow \emptyset$, Test Case Collection @ step 0
    $\Omega[0] \leftarrow \Theta$ = set of not yet satisfied TOs @ step 0
    $\Psi[0] \leftarrow \emptyset$ = set of satisfied test objectives @ step 0
 1: while $((\Psi \neq \Theta) \wedge (l \leq L) \wedge (\neg T_{OUT}))$ do
 2:     $\Omega[l]$ = SelectTestObjectives $(\Omega[l-1], \Theta)$
 3:     $\Psi[l] = \emptyset$
 4:     $bDone = false$
 5:     while $(\Omega[l] \neq \emptyset) \wedge (bDone = false)$ do
 6:         $(\Psi, tc)$ = FindTestCaseMaxSat$(\Omega[l], l)$
 7:         if $(\{tc\} \neq \emptyset)$ then
 8:             $\mathcal{TC}[l] \leftarrow \mathcal{TC}[l-1] \cup \{tc\}$
 9:             $\Psi[l] \leftarrow \Psi[l-1] \cup \Psi$
10:             $\Omega[l] \leftarrow \Omega[l] - \Psi$
11:             $bDone = false$
12:         else
13:             $l = l + 1$
14:             $bDone = true$
15:         end if
16:     end while
17: end while
18: return $\Psi[l], \mathcal{TC}[l]$
```

of the underlying formal back-end. Hence the TO selector can be executed again to select a new subset of TOs or exiting from the loop if no more TOs can be selected. If during last iteration no new test cases are found, it is not possible to satisfy any of the current set of selected TOs for the given length, hence the explored depth bound is increased and a new iteration is started if resource limits are not reached (i.e. timeout, memory consumption, etc.).

1) The algorithm loops until all test objectives are satisfied or a given resource/time/depth bound is reached;
2) While this bound has not been reached:
    a) A subset of not yet satisfied test objectives (TO) is identified. The subset may be the entire set of unsatisfied TOs or it may be driven by the user needs (i.e. all the TOs related to the coverage of a given sub-function of the SUT, etc.)
    b) The following loop will be executed until new test cases are not found or the selection mechanism has not anymore TOs to select:
        i) The formal engine is queried to find all the test cases of length equal to the current length bound that maximize the number of satisfied TOs among the ones selected at previous step. Each test case satisfies a unique subset of unsatisfied TOs i.e. there are not two test cases satisfying the same subset of TOs.
        ii) If new test cases are found, collect all

the subsets of satisfied TOs and remove them from the search to avoid search again their satisfaction in next iteration.

iii) In case no new test cases are found, we can assess that for the given length bound do not exist test cases satisfying the selected test objectives. This claim is possible because of the exhaustive search of the underlying formal back-end. Hence the TO selector can be executed again to select a new subset of TOs or exiting from the loop if no more TOs can be selected.

c. If during last iteration no new test cases are found, it is not possible to satisfy any of the current set of selected TOs for the given length. Hence the explored depth bound is increased

The algorithm is guaranteed to terminate provided that the test objective selection performed at line 2 is able to identify the test objectives previously selected even if not satisfied. The simplest admissible selection mechanism picks the entire set of unsatisfied objectives at once and exits from the outer loop at point 3 after the first execution. More efficient test selection mechanisms are admissible and possible but for brevity we will not cover this topic in the report. During the search at line 6 a sub-procedure is called in order to produce queries to the formal engine and storing the counter-examples provided by the model checker as test cases. A Detailed description of the procedure is provided in Algorithm 2. The overall algorithm proceeds in an incremental fashion. Starting from an initial exploration depth bound all the test cases of a given length that maximally satisfies the test objectives are found. After the formal engine proofs that no more test cases of the given length can satisfy additional TOs, the explored bound is incremented and the search is executed again. Due the exhaustive nature of the search, it is guarantee that the set (or a super-set) of the TOs satisfied by a test case of length k cannot be satisfied by test case of length $j < k$. Hence, the method is efficient in the following sense:

- each new test case covers only new test objectives hence it strictly increments the coverage of the test suite by covering only not yet covered test objectives;
- every test case covers the maximal number of unsatisfied test objectives of a given length

In addition the incremental nature of the algorithm allows for generating test cases by starting from a complex problem in the number of TOs but simpler in the unrolling of the SUT model transition relation and incrementally increasing the complexity due the unrolling of the transition relation but reducing the number of satisfiable test objectives. Our experience with industry sized models shows that this trade off allows for applying ATG on complex models since the satisfied TOs are removed incrementally at every step and the complexity of the formal problem introduced by the increment of the explored bound is partially mitigated by the reduced number of TOs to be satisfied.

B. Maximal Coverage Test Case Generation

The mechanism to find the test case satisfying the maximum number of TOs is an important part of the optimal test suite generation procedure and in this sub-section it will be described in details. The algorithm relies on the concept of monitor variable associated to a test objective TO. A monitor is an integer variable and can be seen as a function m associated to a test objective TO that at every execution step k evolves as follows:

$$m_{TO}[k] = \begin{cases} 1 \quad if \begin{cases} m_{TO}[k-1] = 1, or \\ \exists j \leq k \text{ s.t. } TO \text{ satisfied at step } j \end{cases} \\ 0 \quad \text{otherwise} \end{cases}$$

and

$$m_{TO}[0] = \begin{cases} 1 & \text{if } TO \text{ satisfied at initial step} \\ 0 & \text{otherwise} \end{cases}$$

Each test objective T_k has an associated monitor variable m_k. Given a set of unsatisfied test objectives $T_1, T_2, \ldots, T_N$ and their associated monitors $m_1, m_2, \ldots, m_N$ the algorithm that searches for the test case set is described by the pseudo-code shown by Algorithm 2. The algorithm loops until there are counter examples found (lines $1 \ldots 9$). The search at step 2 searches for a counter example that maximizes the sum of the values of the monitors $m_1, \ldots, m_N$. The counter-example is obtained applying bounded model checking on the input model. The model checker takes into account the dynamics of the SUT and the given coverage metrics (that is used to generate the test objectives and the associated monitors). The maximization procedure ensures that the found counter-example exercises as much test objectives as possible. If a counter example is found the monitor values' configuration is extracted and a new constraint is added to the model in order to exclude the configuration from future searches (line 4, 5). This step ensures the progress of the iteration loop avoiding the model checker to find a counter-example satisfying the same test objectives over and over (there might be an infinity of them). Then the test case is extracted from the counter

Algorithm 2 Maximal Coverage Test Generator Algorithm

Input: model: The enhanced formal model under analysis
Input: $m_1, m_2, \ldots, m_N$: Set of model variables representing unsatisfied test objectives' monitors
Output: testCases : Set of produced test cases

```
1: repeat
2:   find a counter example such that $m_1 + m_2 + \ldots + m_N$ is maximal
3:   if counter example exists then
4:     define $\mathbf{m}^* = [m_1^*, m_2^*, \ldots, m_N^*]$ the found monitor configuration
5:     exclude $\mathbf{m}^*$ from the admissible solutions of the maximization search
6:     extract test case values for the found counter-example
7:     add extracted test case to the testCases set
8:   end if
9: until counter-example has been found
```

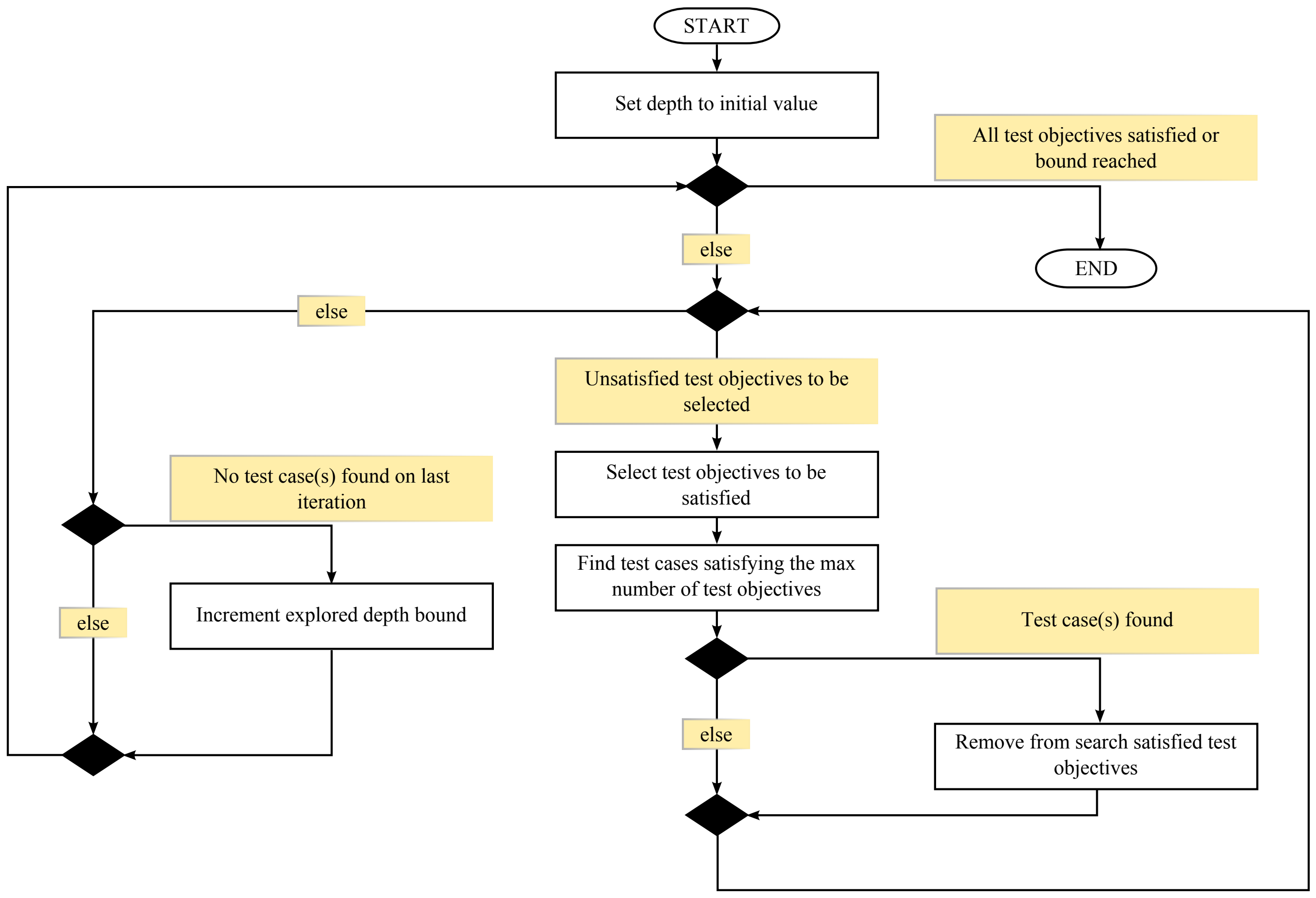

START
Set depth to initial value
All test objectives satisfied or bound reached
END
else
else
Unsatisfied test objectives to be selected
Select test objectives to be satisfied
No test case(s) found on last iteration
Find test cases satisfying the max number of test objectives
Increment explored depth bound
else
Test case(s) found
else
Remove from search satisfied test objectives

Fig. 3. Test Generation Work Flow

example by storing the values of the interesting variables of the system (inputs, outputs and internal variables) and finally it is added to the output set (line 6 and 7). The loop is then executed again looking for additional maximal possible monitor configurations until no more configurations can be found. This guarantees that: 1) every test case produced by the algorithm maximizes the number of test objectives covered and it is not possible that another test case satisfies a super set of the covered ones and 2) for the produced test suite is true by construction that there are not two test cases satisfying the same set of test objectives.

VII. Algorithm Implementation in FSV

Our automatic test generation algorithm has been implemented in the FormalSpecs Verifier (FSV) framework for the verification of embedded systems.

A. *FormalSpecs Verifier*

The FormalSpecs Verifier is a framework targeting complex embedded systems verification. The core of the tool is based on a translator from Simulink modeling language to NuSMV native language. The transformation process produces a semantically equivalent NuSMV representation of the input model taking into account the non-determinism resolution that may be introduced during the transformation step. In Fig. 4 the generic flow is described in details. As a first step the Simulink textual

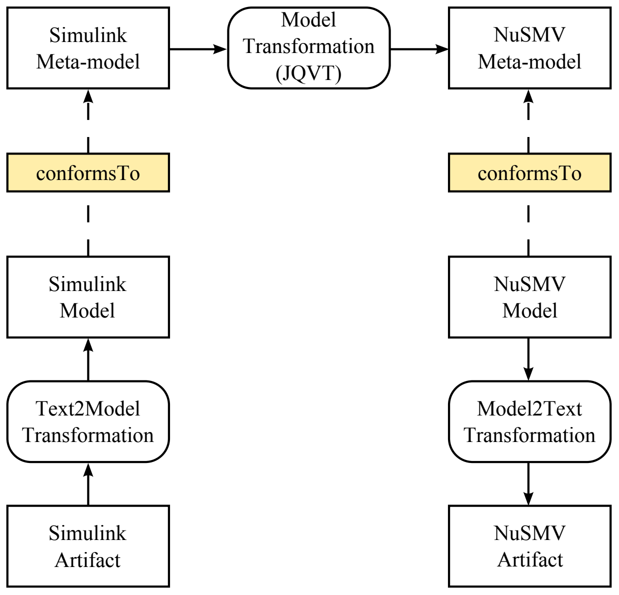


Fig. 4. Formal Specs Verifier Model Transformation Flow

file is parsed. Then the parsed Simulink model is processed generating a semantically equivalent NuSMV model that is used to generate the concrete NuSMV artifact with a model to text step. The technology used to perform the model trans-

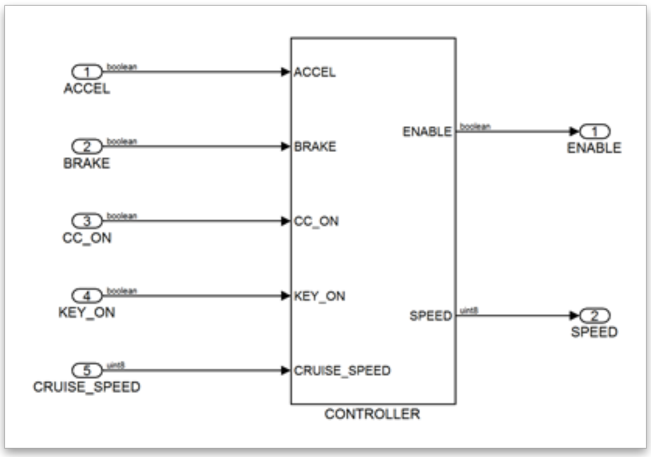


Fig. 5. Controller sub-system

formation step is an internally developed Java embodiment of the OMG Query/View/Transformation (QVT) language called JQVT. The JQVT library aims at providing an industry-level operational implementation of the QVT language. It supports the definition of QVT mappings and the definition of mappings inheritance, disjunction and merging. JQVT allows capturing the mapping relation that links a source model element to a target model element and it supports the *resolve* and *resolveIn* operators to retrieve the set of mapping source model elements from a given mapped target model element. JQVT does not support the entire QVT specification. However, it has been extensively used as translation infrastructure of different tools for the translation of industry-level sized models [12].

B. Cruise Control Example

We show the application of our test generation algorithm to a cruise control reference example implemented in MATLAB Simulink. We consider a modified version of the model proposed by Aldrich in [13]. A cruise control is an embedded system that regulates the speed of a vehicle based on a set of commands provided by the driver; the interface of the control algorithm appears to the system as represented in Fig. 5. The ACCEL and BRAKE inputs are Boolean values representing the pressure of the accelerator and brake pedal by the vehicle driver. The CC_ON Boolean input is set when the driver wants to engage the cruise control. The KEY_ON input represents the presence of the key (Boolean value) and finally the CRUISE_SPEED value is an unsigned integer input set by the user representing the desired cruise speed. The outputs of the controller are the ENABLE Boolean value that is true whenever the cruise controller is actively controlling the vehicles speed and the SPEED unsigned integer value representing the reference speed passed to the cascade speed controller that acts directly on the engine throttle based on the reference and current speed values. In Fig. 6 the internal modal logic of the controller is represented as an extended finite state machine (Simulink Stateflow machine).
The controller is initially in an OFF state and passing thru an IDLE state can go on a controller active mode (CC_MODE) or disengaged mode (CC_MODE_DISABLED) in which it returns the direct control to the driver. This happens whenever

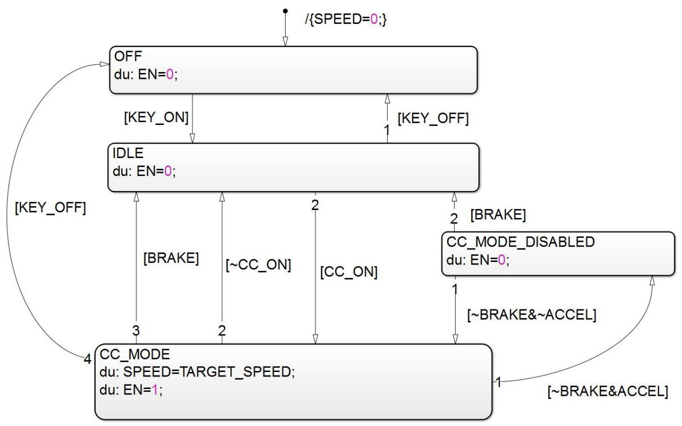


Fig. 6. Controller Extended Finite State Machine

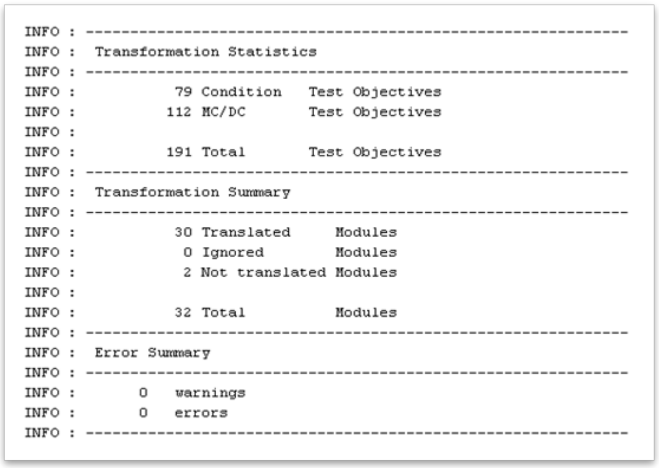

```
INFO : -------------------------------------------------------------
INFO :  Transformation Statistics
INFO : -------------------------------------------------------------
INFO :          79 Condition   Test Objectives
INFO :         112 MC/DC       Test Objectives
INFO :
INFO :         191 Total       Test Objectives
INFO : -------------------------------------------------------------
INFO :  Transformation Summary
INFO : -------------------------------------------------------------
INFO :          30 Translated     Modules
INFO :           0 Ignored        Modules
INFO :           2 Not translated Modules
INFO :
INFO :          32 Total          Modules
INFO : -------------------------------------------------------------
INFO :  Error Summary
INFO : -------------------------------------------------------------
INFO :       0   warnings
INFO :       0   errors
INFO : -------------------------------------------------------------
```

Fig. 7. Automatic Generation of Test Objectives Results

she/he interacts with the car pressing the acceleration or brake pedal. The FormalSpecs Verifier Automatic Test Generation (FSV-ATG) tool is executed to perform the efficient generation of test cases. As a first step the input model is elaborated and automatically enriched with a set of test objectives to enable the subsequent test generation procedure for covering the state machine to obtain MC/DC coverage. The summary of the generation process for the cruise control example is shown in Fig. 7: a total number of 191 test objectives have been added to the formal model to enable the ATG step. The ATG activity is executed and as results generates a set of efficient test case. For the cruise control example the ATG generated 31 test cases given a bound of 100 maximum test cases and a depth bound of 30 steps. To validate the obtained results we simulated the generated test cases on the SUT evaluating the coverage value using the Simulink Verifcation and Validation (V&V) toolbox which provides an independent measure of effectiveness for our approach. The obtained independent measured coverage has been of 100% for decision, condition coverage and MC/DC coverage.

C. Additional experiments

In order to quantitatively compare the proposed algorithm and implementation with respect to state-of-the art tool we

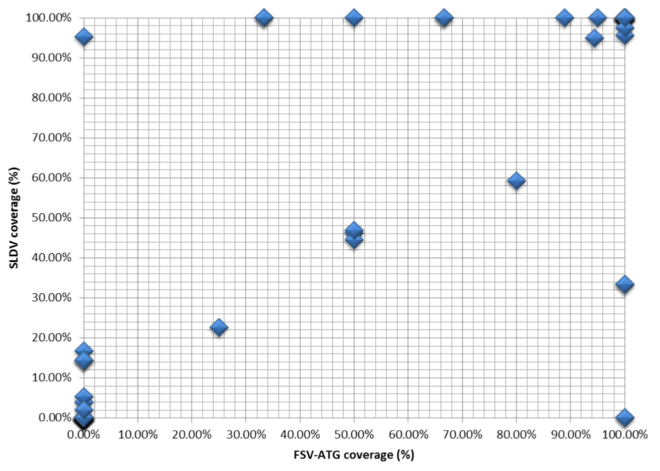


Fig. 8. Comparison of MCDC Coverage Results

compared the test suites generated using the Formal Specs Verifier ATG tool with the Simulink Design Verifier tool (SLDV) [2] that represents an industrial strength tool for the generation of test suites from MATLAB Simulink/Stateflow systems. A set of verification cases has been set up containing blocks ranging over a rich subset of Simulink/Stateflow Libraries. The results for the comparison of the MCDC coverage of the produced suites are presented in Fig. 8. The coverage values are obtained using the MATLAB VnV toolbox[3]. The analysis of the results shows that in some cases the SLDV tool is capable of achieving higher coverage with respect to FSV-ATG whereas in other cases the opposite is true. Cases where the MC/DC coverage percentage is 0% for FSV-ATG indicate that our algorithm was not able to generate any test cases. The analysis of the "losing" cases for the FSV-ATG tool showed that the low coverage is due to an inefficient translation of the Simulink/Stateflow block that does not allow efficient application of our ATG algorithm, as highlighted in Sec. VII-A. As an example, in some cases the FSV-ATG tool produces a structure of comparison/logical blocks that is inefficient for generating a high number of test objectives. The optimization of the translation step for achieving efficient coupling with the test generation engine is one of the activities we have planned as the next development steps of our tool.

VIII. Conclusion

Our model-based test generation algorithm produces a test suite starting from a model of the system under test (SUT) that is enriched with a set of test objectives to be satisfied by the test cases derived from a given coverage metric. The produced suite covers the model test objectives in an efficient way such that 1) there are no two test cases satisfying the same set of test objectives and 2) each test case covers the maximum number of test objectives for a given length. Our algorithm relies on the combination of bounded model checking with an optimization-based formulation of the test generation problem. The algorithm is implemented using an incremental execution approach that mitigates the complexity of the problem allowing successful application to complex industrial use cases. Several ways of algorithm-improvement are possible, though. From a technical standpoint the algorithm could be improved by employing parallelism at test generation level and not only at the formal back-end level. In addition, the last advancements in pseudo-Boolean SAT solving can be taken into account to explore additional formal back-ends. From a methodological standpoint the use of contract-based design ([14], [15]) could allow for exploiting a compositional approach at test generation level. From a methodological standpoint the use of contract-based design could allow for exploiting a compositional approach at test generation level. Finally, the use of formal proofs on the model can be used to ease the process of test generation.

[2]https://www.mathworks.com/simulinkdv

[3]http://www.mathworks.com/products/simverification/

References

[1] O. Ferrante, A. Ferrari, and M. Marazza, "Model based generation of high coverage test suites for embedded systems," in *European Test Symposium*, 2014.

[2] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *Software Engineering, IEEE Transactions on*, vol. 36, no. 2, pp. 226–247, 2010.

[3] S. Rayadurgam and M. P. E. Heimdahl, "Generating mc/dc adequate test sequences through model checking." in *SEW*. IEEE Computer Society, 2003, p. 91.

[4] O. Ferrante, L. Benvenuti, L. Mangeruca, C. Sofronis, and A. Ferrari, "Parallel nusmv: A nusmv extension for the verification of complex embedded systems," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, F. Ortmeier and P. Daniel, Eds. Springer Berlin Heidelberg, 2012, vol. 7613, pp. 409–416.

[5] A. Gargantini and C. Heitmeyer, "Using model checking to generate tests from requirements specifications," in *Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. ESEC/FSE-7. London, UK, UK: Springer-Verlag, 1999, pp. 146–162.

[6] M. Kadono, T. Tsuchiya, and T. Kikuno, "Using the nusmv model checker for test generation from statecharts," in *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on*, 2009, pp. 37–42.

[7] G. Di Guglielmo, F. Fummi, G. Pravadelli, S. Soffia, and M. Roveri, "Semi-formal functional verification by efsm traversing via nusmv," in *High Level Design Validation and Test Workshop (HLDVT), 2010 IEEE International*, 2010, pp. 58–65.

[8] G. Hamon, L. deMoura, and J. Rushby, "Generating efficient test sets with a model checker," in *2nd International Conference on Software Engineering and Formal Methods*. Beijing, China: IEEE Computer Society, Sep. 2004, pp. 261–270.

[9] http://www.mathworks.com/products/simulink/.

[10] J. Chilenski and S. Miller, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, vol. 9, no. 5, pp. 193–200, 1994.

[11] M. Marazza, O. Ferrante, and A. Ferrari, "Automatic generation of failure scenarios for SoC," *ERTS*, 2014, February 5th.

[12] A. Ferrari, L. Mangeruca, O. Ferrante, and A. Mignogna, "DesyreML: a sysml profile for heterogeneous embedded systems," *ERTS, Embedded Real Time Software and Systems*, 2012.

[13] W. Aldrich, "Coverage analysis for model based design tools," *Proc. of the 18th International Conference and Exposition on Testing*, 2001.

[14] L. Mangeruca, O. Ferrante, and A. Ferrari, "Formalization and completeness of evolving requirements using contracts," in *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, 2013, pp. 120–129.

[15] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for System Design," INRIA, Rapport de recherche RR-8147, Nov. 2012. [Online]. Available: http://hal.inria.fr/hal-00757488

Model Checking of SCADE Designed Systems

S. Heim[1], X. Dumas[1], E. Bonnafous[1]

P. Dhaussy[2], C. Teodorov[2], L. Leroux[2]

1: CSSI, 3 rue du professeur Pierre Vellas, Toulouse, France
2: ENSTA-Bretagne, Lab-STICC UMR CNRS 6285, Brest, France

Abstract

Keywords: model checking, formal methods, CDL, SCADE, LUSTRE, OBP, synchronous, asynchronous

Introduction

Model checking **[1]** is a well-known method to verify a formal model in all possible configurations. Nevertheless this technique can hardly scale up to industrial **asynchronous** systems because of the **state-space explosion problem [17]**.

To address this challenge, a new approach based on **context specification** (the environment of the system) and an observation engine called **OBP** (Observer Based Prover) has been developed **[2]**. The idea is that given a property to be verified, one doesn't need to explore all possible configurations of the complete system. Among all **possible behavior** of the system, a tiny part is representative enough for the property to be verified.

Thus, specifying a pertinent environment (a context) allows **restricting the system behavior** on those only parts where the property is worth verifying.

The objective of our work is to apply this Context-aware verification method to the verification of **SCADE [3, 10]** systems designed in LUSTRE language, in order to check **behavioral properties** related to system safety.

Moreover **LUSTRE [4, 9]** is a **synchronous** language whereas OBP exploration engine takes as input an asynchronous model designed in **FIACRE [5]** language.

To cope with this problem our approach consists in developing a **GALS** method combining asynchronous contexts with synchronous models **[6, 7, 8]**.

The interest of our new approach is twofold:

- Verifying formal properties on synchronous industrial systems with formal methods using GALS approach,
- Facing the state-space explosion via context aware specification;

To our knowledge, there's no work combining those two previous methods.

This document is organized as follows:

First, a state of art on existing methods combining synchronous system modeling within an asynchronous environment is presented.

Next, we expose the **GALS** methodology approach we combined with context aware verification method.

Then we introduce two case studies used for experimentation of our method.

Eventually we conclude and present some perspectives for future work.

This work is done in the frame of the French R&D project **DEPARTS** ([1]), which is a **FSN/BGLE** project supported by **BPI France**.

1. State of the art

Mixing synchronous and asynchronous model designs. As demonstrated into previous studies (cf. Airbus **[12],** and Rockwell-Collins **[13]**), using an explicit model-checker for synchronous language verification may be required to verify some asynchronous properties between synchronous parts of a system.

Moreover, traditional "synchronous-observers" verification approach is not applicable in that case: asynchronous behaviors' modeling is not possible with synchronous language (i.e. communication delays, asynchronous clocks between processors).

GALS. Verifying a synchronous system in an asynchronous environment is not an easy task because of synchronicity assumption: Input/Output computations are considered to take no time.

To cope with this problem several GALS approaches have been developed **[6, 7, 8]**.

([1]) DEsign PAtterns for Real Time and Safe applications.
FSN - Fonds national pour la Société Numérique.
BGLE - Briques Génériques du Logiciel Embarqué.

In **[7]**, the work consists in generating C code from the synchronous language **SIGNAL [14]**. Then, this code is called atomically (to ensure synchronicity assumption) in an asynchronous formal language: **PROMELA [15]**. The environment closing the overall system is then designed in PROMELA and the verification of a property is done with **SPIN** model checker **[15]**.

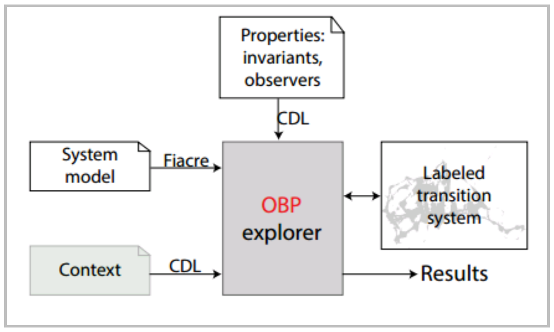
Nevertheless when dealing with huge systems, the environment grows drastically generating a state-space explosion. For this reason, an optimized exploration method has been developed based on OBP explorer and its associated Context Description Language called CDL [2].

Context-aware Verification approach. State-space explosion is intrinsically related on the way model checking method works. Model checking consists in **closing** a formal system with all possible behaviors of its environment, and then exhaustively analyzing the emerging executions. The idea behind the Context-aware Verification methodology **[2]** is that only a subset of the environment is necessary in accordance with the property one want to verify. This explicit description of the environment has many benefits:

- The environment can be decomposed by **several contexts** focusing on different system modes.
- When the environment is **too large**, it can be **decomposed** by OBP (splitting method **[11]**) to generate **independent sub-contexts, which are** successively composed with the system and the property so that to make several little verification.
- By enforcing some structural properties on the environment behaviors, OBP explorer can also use **optimized algorithms** such as PastFree[ze] to reduce the verification time **[16]**.
- Properties are verified only on specific context definitions.

The following picture could summarize the Context-aware Verification approach, implemented in the OBP toolkit.

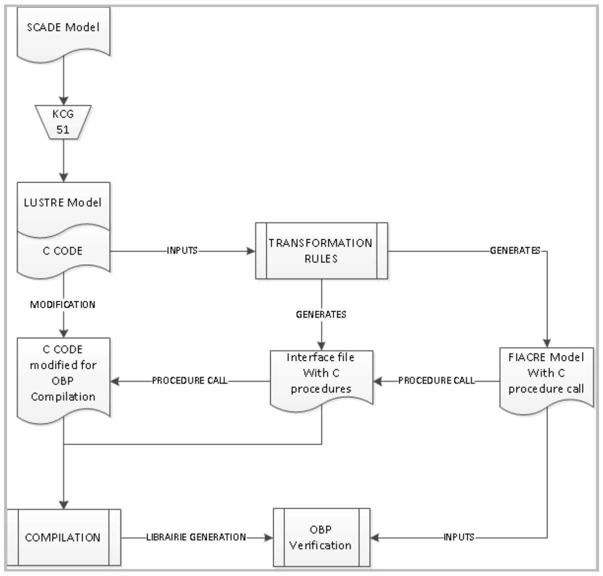
[2] www.obpcdl.org



2. Contribution

In this paper we propose to combine a GALS verification methodology with the Context-aware Verification approach. Our technique uses the synchronous LUSTRE language designed with SCADE tool and FIACRE asynchronous language used by OBP. We have experimentally validated our approach using two realistic case studies from the automotive and aero-space domain. In this study we focused on the verification of functional properties. Nevertheless our approach could integrate other classes of properties, and can accommodate techniques for guaranteeing the numerical accuracy.

The following picture summarizes our method.



Our approach is structured as follows:

1. We first design the system with SCADE components based on LUSTRE language;
2. Then we generate C code from the LUSTRE model thanks to the qualified SCADE code generator KCG51;

3. Next, from C code we generate the corresponding FIACRE model in accordance with the synchronicity assumption;
4. To make possible the compilation of C code, we generate some wrappers; the wrapper is useful to exchange data between FIACRE model and C code called functions;
5. FIACRE system with C code called function is generated so that to make the OBP exploration possible;

Once the FIACRE system is generated, following the previous five 5 steps one can implement an environment (an OBP context) to verify properties.

Nevertheless, some stimuli sent from the environment needs some parameters values so that to make the system under verification evolving.

To this purpose, we choose to generate a data structure into the FIACRE system containing all the input and output values which can be exchanged with the environment.

The data structure contained in the FIACRE model will be used for verification purpose by OBP tool.

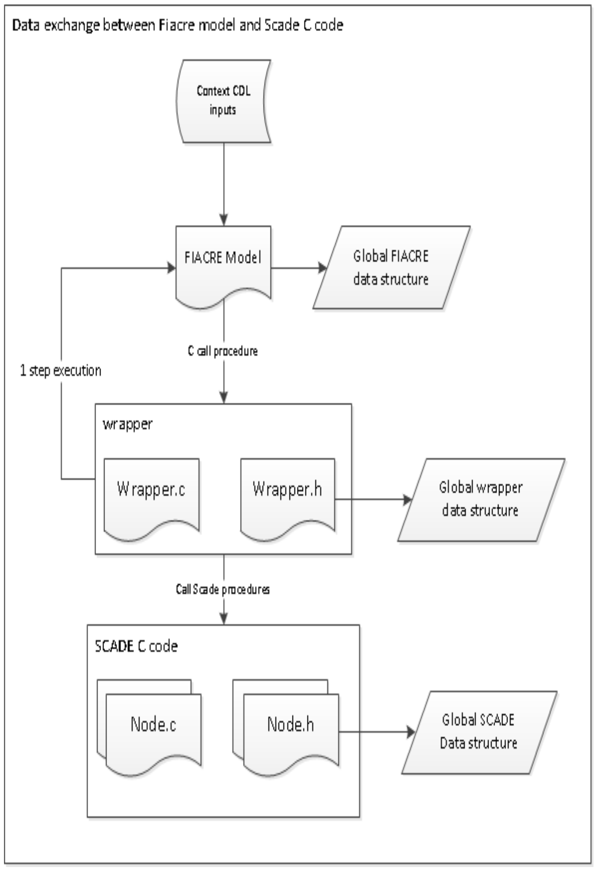
This implementation is well suited to our methodology because SCADE system stores all input and output values on global data structure too.

The stimuli and parameters sent by the environment are therefore copied into the FIACRE data structure (identical to SCADE C code data structure).

This data structure is passed in C call function parameters which are dedicated to the computations and modifications of input output values of the system.

When the C call function has ended, the data structure which has been modified is copied again into the FIACRE data structure so that OBP tool could display and verify properties from this FIACRE data structure.

The following picture summarizes the data exchanges between the environment, the FIACRE system and C call procedures.


Data exchange between Fiacre model and Scade C code
Context CDL inputs
FIACRE Model
Global FIACRE data structure
C call procedure
1 step execution
wrapper
Wrapper.c
Wrapper.h
Global wrapper data structure
Call Scade procedures
SCADE C code
Node.c
Node.h
Global SCADE Data structure

3. Environment Modeling and Analysis

3.1 Environment modeling

In the case of Context-aware Verification, the environment modeling should be seen as a **methodological phase** that needs to balance two important constraints while building the context.

First the context has to cover enough behaviors to be considered valid for a given property. But at the same time it has to be small enough to be possible to exhaustively explore the product of its composition with the system under study (SUS).

The context is modelled starting from the **system requirements** one want to verify. From the requirements analysis, the designer identifies all the actors of the system that can interact and send some stimuli to the system.

For each actor, its behavior is refined by describing possible actions it can send to the system.

Eventually all the actors behaviors' are interleaved so that to generate all possible scenario of the environment.

Of course the more actors the worst, because each actor behavior is interleaved with all others potentially generating a combinatorial explosion of the environment state space during unfolding and interleaving step.

This phase is done manually, and relies on the engineering judgment.

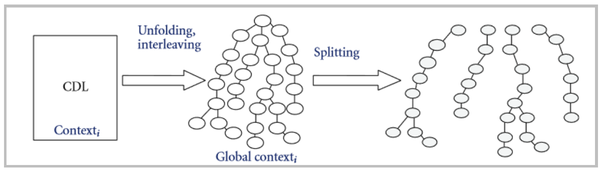
To face this environment state space explosion, a **pertinent modelling** of the environment must be described **by the engineer** who will for instance:

- discard some actors with no relation to the requirements,
- discard some useless actors stimuli with regard to the requirements he wants to verify,
- create several different environment with relation to the set of requirements he wants to verify,
- create a specific initialization sequence events to dig the system in a pertinent state for the verification.

3.2 Environment Analysis

Nevertheless a pertinent environment modeling is not always sufficient to face the environment combinatorial explosion.

For this purpose, the context aware verification tool (OBP) incorporates some algorithm to reduce environment state space explosion thanks to the splitting method.



The splitting method consists in decomposing the global context generated by the interleaving of all the actors' events in a set of global "sub-contexts" which will be composed with the system under study.

This method allows verifying systems stimulated by a complex environment, and covers exhaustively the whole generated state-space.

The combinatorial explosion environment behavior in space due to environment is transferred to combinatorial explosion in time due to the countless "sub-contexts" generated.

Nevertheless this new combinatorial explosion can be faced by parallel verification of the "sub-contexts" distributed to a set of machines.

An important observation is that while with the **automatic-split technique** the state-space is decomposed in several partitions, these partitions are not disjoint.

Hence the sum of these explorations with splitting represents the analysis of nearly two times more states and transitions than the exact initial state-space without splitting.

Nevertheless, we believe that this is a small price to pay for the possibility of analyzing five times larger state-space without the need of doubling the physical memory of the machine.

4. Case Studies

4.1 Roll-Control. We have first applied our method to a simple case study: a **Roll-Control** system.

The Roll-Control system allows to compute the Roll-Rate value, and to generate roll warnings whenever the roll rate is greater than 15° or lower than -15°.

The environment of this system is composed by three « actors »:

- Pilot actions on joystick
- Left and Right yaw applied on the plane

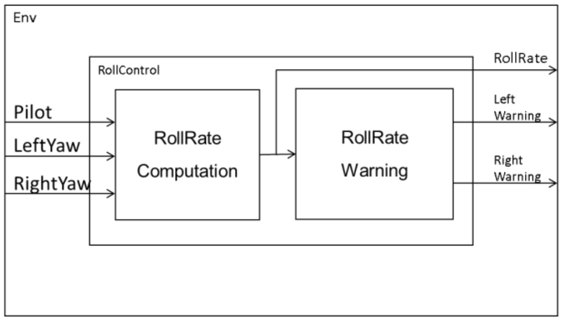
As a result of those inputs, the Roll-Rate is updated and warnings are activated in case the Roll-Rate is out of range.

First, the simplified coupling effect is calculated, then, the plane roll rate is calculated as follows:

rollCoupling = (leftAdvYaw – rightAdvYaw) × 0.1
rollRate = (joystickCmd – rollCoupling) × 0.25

The absolute value of the Roll Rate has to be saturated to 25.0.

The Roll-Control system is described in the following figure 4.1.



The Roll-Control is composed of 2500 lines of C code.

We have successfully checked following property on this model with OBP exploration engine:

The roll-control system shall never raise "left roll warning" and "right roll warning" at the same time;

This case study has successfully passed verification steps, because of its small size and limited possible behaviors.

4.2 Cruise-Control system. We then applied our method on an automotive Cruise-Control System (CCS) designed in SCADE.

This section provides an overview and some requirements of this case study.

Functional Overview. The CCS main function is to adjust the speed of a vehicle.

After powering the system on, the driver first has to capture a target speed, and then it is possible to engage the system. This target speed can be increased or decreased by 5 km/h with the tap of a button.

There are also several important safety features. The system shall disengage as soon as the driver hits the brake pedal or if the current vehicle speed (S) is out of bounds (40 < S < 180 km/h). In such case, it shall not engage again until the driver hits a "resume" button. If the driver presses the accelerator, the system shall pause itself until the pedal gets released.

Architecture overview (cf. Figure 4.2).

The CCS is composed of 3 mains parts: a "control panel", a "system center", and an "actuation manager" (for speed and throttle calculation).

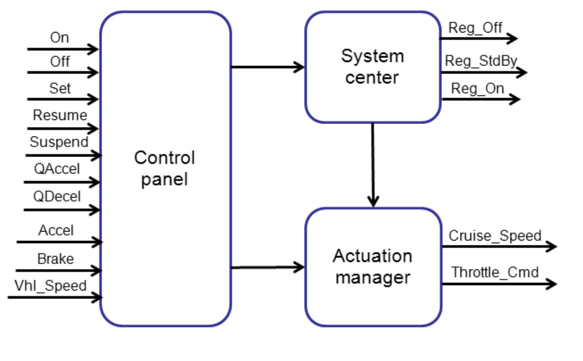


Fig. 4.2 – Cruise Control Architecture

The **control panel** is in charge of converting inputs signals from user to provide them to the system.

The **actuation manager** is able to capture the current speed and, once enabled, to adjust the vehicle speed to the defined target speed (and also throttle command value).

The **system center** component, that acts as a controller, and includes a state machine (states OFF, STDBY, ON).

The control panel acquires signals following from buttons used by the driver to operate the system:

- On, Off: Enable or disable the system
- Set: Capture the current speed as the target value
- Resume: Engage the control speed function
- Suspend: Disengage the control speed function
- QuickAccel: Increase the target speed by step
- QuickDecel: Decrease the target speed by step

In our case, the "control panel" is also responsible of providing BrakePressed and AccelPressed signals, which are built with Brake and Accel pedals signals, and to compute SpeedOutOfBounds signal. All are booleans signals provided to system center, with behaviours defined below:

- Brake pedal pressed: induces disengagement,
- Speed of the vehicle goes out of bounds: induces disengagement,
- Accelerator pedal pressed or released: pauses or resumes the speed control function;

The actuation module provides means for the system to interact with the vehicle. It can capture the current speed of the vehicle, and use it as a new target speed value. Once the CCS enabled, the actuation is responsible for controlling the vehicle speed accordingly.

Finally, the system center is the core of the CCS. It is responsible for handling events detected by control panel module.

Then system center use these events to switch to right system state, and to engage control function or not:

- From OFF to STD_BY: on btn_On,
- From STD_BY to ON: on btn_Set,
- From ON to STD_BY: on btn_Suspend or Brake,
- From STD_BY to OFF: on btn_Off;

Requirements. This section lists three main requirements of the CCS system and shows how to model them using the CDL formalism, with predicates or observers automatons.

REQ.1: The system shall not engage itself if the target cruise speed is not set.

REQ.2: The target Speed shall never be lower than 40 km/h or higher than 180 km/h.

REQ.3: When the system is powered off, the target speed shall be reset, and considered as unset.

REQ.1 can be encoded by using an **observer automaton**, on figure 4.3 below. To encode this observer using CDL formalism, we first need to introduce the events triggering the transitions

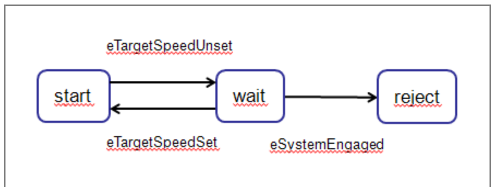


Fig. 4.3 – observer automaton for REQ.1

4.4	**predicate** pCruiseSpeedIsUnset is {
	{sys}1:context._Cruise_speed < 40
	or {sys}1:context._Cruise_speed >180 }
	event eTargetSpeedUnset is {
	pCruiseSpeedIsUnset **becomes** true }

On listing 4.4, pCruiseSpeedIsUnset is a predicate on cruise speed value, read from interface structure (context) of the main process {sys}, returning true if the constraint is verified.

Then an event eTargetSpeedUnset is built with “becomes true” formula in order to express a rising edge of the predicate, which is an **observable event** in OBP observation engine.

On listing 4.5, another event can be defined based on system center state machine output value (Regul_ON).

4.5	**predicate** pSystemIsEngaged is {
	{sys}1:context._Regul_ON = 1 }
	event eSystemEngaged is {
	pSystemIsEngaged becomes true }

Using these events, the observer automaton of figure is defined in listing 4.6:

4.6	**property** REQ1 is {
	start **--** eTargetSpeedUnset **-->** wait;
	wait **--** eSystemEngaged **-->** reject;
	wait **--** eTargetSpeedSet **-->** start }

We can then encode two others requirements REQ.2 and REQ.3 using the same principles.

To model predicates and events, we could also use internal states of concurrent processes of the system.

Environment. In the case of the CCS the environment is built from two main distinct actors modeling:

a) a nominal scenario,

b) a disruptor;

The basic scenario can be seen as a linear use case of the CCS that covers all the functionality involved by the properties we aim to verify.

This scenario must pass through following steps:

Event	Behavior
Press Accelerate pedal	Vehicle speed grows
Button On	CCS ON / stand-by
Button Set	Target speed set
Stop Accelerate pedal	CCS engaged / regulate
Brake pedal	CCS in stand-by
Resume button	CCS engaged
Button Off	CCS OFF

The disruptor is a wide alternative including changes of the vehicle speed within the allowed range or not, pressure on the pedals and the panel buttons. The disruptor stresses the SUS against a number of possible unexpected behaviors of the environment.

The disruptor encoding refers to verification environment capability of sending events to system, including speed target requests, pressure on pedals, or on panel buttons.

4.7	**activity** disruptor is {
	eRegularSpeed_v1/v2 [] eAbnormalSpeed_v3/v4
	[] ePressPedals_p1/p2 [] ePushButton_b1/b2/b3 }

Once these two actors are composed asynchronously, we get a wide range of variations of the basic scenario using the capabilities of the disruptor at all stages.

4.8	**Cdl** myContext is {
	Properties req1 Assert req2, req3
	Init is { eBtnOn }

	Main is { basic_scenario
	|| disruptor }
	}

4.3 Verification Results

This section presents the results obtained for the verification of the main requirements previously presented, emphasizing the importance of the Context-aware Verification approach, applied on our GALS approach.

During the exploration, we have tried several kinds of observers, and we have intentionally create errors into the model in order to check that there were detected (assertion, or reject state of automaton).

The verification results for the CCS case study are:

- 17.847 states and 62.771 transitions (with 3 processes, in 21 sec);
- 367.800 states and 1.621.000 transitions (with 4 processes, in 571 sec)

So we can conclude that our approach does not produce a too large state space, due to encapsulation technique used (atomic execution of synchronous function into asynchronous process).

Even if we have not yet modeled all required behaviors as asynchronous communication between several more synchronous processes, it seems to be a very promising approach for larger and complex systems.

As a comparison, our partner from **Lab-STICC** has applied same verification context on an asynchronous CCS UML model, which generates 3 millions of states and 10 millions of transitions (with 4 processes, and 4 ticks of clock only) (cf. **[18]**).

For the moment, we have only used traditional Breadth-First Search (BFS) reachability algorithms. But we know that in case of a larger state space, we could also use PastFree[ze] algorithm and splitting technique.

The use of the PastFree[ze] algorithm enable the analysis of a 2.4 times larger state-space, and the joint use of PastFree[ze] and automatic split technique enable 4.78 times larger state-space, compared to traditional Breadth-First Search (BFS) reachability algorithms, without the need of extending the physical memory of the machine.

5. Conclusion and Perspectives

In this paper we have used the Context-aware Verification technique for the analysis of several requirements of a Cruise-Control System composed of synchronous languages functions.

Modeling and verification of asynchronous properties of this kind of systems based on composition of synchronous components, renders traditional synchronous model-checking approaches inefficient.

Using the environment reification through the CDL formalism, this task becomes manageable by relying on two powerful optimization strategies.
These strategies rely on the structural properties of the CDL contexts and enable the reachability analysis of larger industrial models.

While the approach presented in this paper offers promising results, for this technique to be used on industrial-scale critical systems, some work has to be done on the formalization of the context coverage, with respect to the full-system behavior, in order to assist the user on initial context specification.

References

[1] Clarke, E.M., Emerson, E.A., Sistla, A.P.: "Automatic verification of finite-state concurrent

systems using temporal logic specifications", ACM Trans. Program. Lang. Syst., vol. 8, p. 244--263, ACM, New York USA, 1986.

[2] Dhaussy P., Boniol F., Roger J.C.: Reducing state explosion with context modeling for model-checking. In: 13th IEEE International High Assurance Systems. Engineering Symposium (Hase'11). Boca Raton, USA (2011)

[3] "SCADE 6: A Model Based Solution For Safety Critical Software Development", François-Xavier Dormoy. ERTS 2008, Esterel Technologies;

[4] "The Synchronous Dataflow Programming Language Lustre", N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, Proceedings of the IEEE, 79(9):1305-1320, September 1991.

[5] B. Berthomieu, J. P. Bodeveix, M. Filali, H. Garavel, F. Lang, F. Peres, R. Saad, J. Stoecker, F. Vernadat. The syntax and semantics of FIACRE v2, specification, 2009.

[6] H. Garavel and D. Thivolle. Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In Proc. of SPIN, pages 241–260, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] F. Doucet, M. Menarini, I. H. Kruger, R. Gupta, and J. P. Talpin. A Verification Approach for GALS Integration of Synchronous Components. Electron. Notes Theor. Comput. Sci., 146:105–131, January 2006.

[8] H. Günther, S. Milius, O. Möller: On the formal verification of systems of synchronous software components, VerSyKo project, 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP, MBEES), 2012.

[9] "The Foundations of Esterel", Gérard Berry. In "Proofs, Languages, and Interaction, Essays in Honour of R. Milner," G. Plotkin, C. Stirling, and M. Tofteed., MIT Press (2000).

[10] Methodology Handbook Efficient Development of Safe Avionics Software with DO-178B Objectives Using SCADE, 2011;

[11] Dhaussy, P., Boniol, F., Roger, J.C., Leroux, L.: Improving model checking with context modelling. Advances in Software Engineering ID 547157, 13 pages (2012)

[12] "Model-checking Flight Control Systems : the Airbus experience", ICSE Companion, pages 18–27, T.Bochot (Airbus), V.Wiels (ONERA), P. Virelizier, H. Waeselynck, 2009.

[13] Choi, Y., "From NuSMV to SPIN: Experiences with model checking flight guidance systems", Formal Methods in System Design 30 (FMSD) , 199-216 (2007)

[14] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming Real-Time Applications with SIGNAL. Proceedings of the IEEE, 79(9): 1321-1336, September 1991.

[15] Holzmann, G. J., The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, 2004. ISBN 0-321-22862-6.

[16] C.Teodorov, L.Leroux, P.Dhaussy: Context-aware Verification of a Cruise-Control System, In Proceedings of Model and Data Engineering 4th International Conference, MEDI 2014, LNCS 8748;

[17] Valmari, A.: The state explosion problem, in Springer LNCS volume 1491, 1998, pp 429-528 (http://dx.doi.org/10.1007/3-540-65306-6_21)

[18] P. Dhaussy, L. Le Roux, et C. Teodorov : Vérification Formelle de Propriétés, application au cas d'étude CCS en UML, dans le cadre du projet DEPARTS, Revue Génie Logiciel, n° 109, juin 2014

Industrial Grade Model Checking
Use Cases, Constraints, Tools and Applications

Mathieu Clabaut[1], Ning Ge[2,⋆], Nicolas Breton[1], Eric Jenn[2,⋆⋆], Rémi Delmas[3], and Yoann Fonteneau[1]

[1] Systerel, Aix-en-Provence — Toulouse, France `firstname.name@systerel.fr`
[2] IRT Saint-Exupéry, Toulouse, France `firstname.name@irt-saintexupery.com`
[3] ONERA, Toulouse, France `remi.delmas@onera.fr`

Abstract. *Model checking* has made a lot of progress since its infancy. For a long time, industrial applications were still limited to some very specific domains out of which the technique bumps into the state explosion wall. Nowadays things evolve and some tools are able to tackle real world use cases outside of the known domains.
We give here the feedback collected when using *model checking* on several industrial strength use cases and give indication on how we take into account the specific domain constraints.

1 Model Checking for Industrial Problems

Model checking refers to the problem of exhaustively and automatically checking whether a given model of a system meets a given specification.

Model Checking is now an old technique which takes its ground in the mid 1970s as a response to concurrent problem analysis. It was until recently essentially confined to some specific areas, such as hardware analysis or protocol verification. Extension to other domains such as software verification has always been difficult due to the combinatorial explosion problem (the size of the space state grows exponentially with the size of the problem to analyze).

However, recent developments in a variety of fields, ranging from symbolic *model checking* to SAT solver engines and including *model checker* parallelization lead to a broader range of application in industry including software analysis.

⋆ Seconded from Systerel, Toulouse, France
⋆⋆ Seconded from Thales Avionics, Toulouse, France

1.1 Industrial Use Cases

Model checking may be used for the following use cases, depending on the tool abilities:

Safety Proof consists in verifying some properties on a system model. The model may be designed by hand or automatically derived from existing artefacts, such as `Ada`, `C`, `Simulink` or `Scade` code...
If one of the properties is not being verified, the tool provides a counterexample explaining why the property does not hold.

System Debugging. One of the great features of *model checking* is its ability to provide the user with counterexamples. Such a feature may be used in different ways, but is generally of a good help for debugging a system.
For example, when designing a complex system which should ensure a safety property (e.g. a non collision property for an automatic train system), it is very useful to debug the root concepts which should ensure the safety upon a system model on which you can analyse counterexamples.

Equivalence Checking. The ability to prove some properties upon an existing model

may be used as a way to prove the equivalence of two models.
One application of *equivalence checking* is to verify that a software design in `C` satisfies its specification, by proving the equivalence between a model of the design and a manually written model of the specification.
Another application is to verify the correctness of a tool transformation — for example a translation from one formalism to another — by proving the equivalence between the before and after transformation artefacts.

Constraint Solving. A somewhat more contrived use of *model checking* is the use of its ability to provide counterexamples as a way to solve a constraint problem. The way to do it is to ask the tool to verify that the problem has no solution. If one exists, the tool will return a counterexample which is one solution to the problem.

Test Case Generation. Another use case, derived from the preceding one, is to use a *test objective* as a constraint. The provided counterexamples then give inputs and associated oracles which fulfill these test objectives and may be used as test scenarios.

1.2 Industrial Constraints

Putting aside the many use cases listed before, industry faces many constraints which may have prevented the use of model checking until now:

Regulatory Constraints Many companies must obtain approval from a suitable authority that the system they develop is acceptably safe to operate with regards to the applicable assurance standards (CENELEC EN-50128, DO-178C,…).
As such, it must be shown that the tools who have contributed to the system or to its verification have been qualified with respect to their usage and to their contribution to the global safety. Such a qualification bears a lot of constraints upon the tools and their development process. Few of the known *model checkers* are designed with such compliance in mind.

Cost Reduction The use of formal methods and *model checking* is of interest for industry only if they lead to cost reduction or to standard compliance. In a context where standard compliance is already achieved, the only motivation left for applying formal methods is to gain significantly over costs. Such an objective may be reached either by using *model checking* in order to automate some testing steps or in a less measurable way, by rising the overall quality beyond what is required by the regulatory constraints. Proving a property is indeed really an improvement over testing it, even in the frame of standard compliance, and can lead to finding bugs that would be otherwise discovered much later and would cost a lot more to be corrected.

2 The S3 Toolbox

2.1 S3

Systerel Smart Solver[4] (S3) is Systerel's response to the aforementioned use cases and industrial constraints. S3 is constructed around:

– a high level synchronous modelling language,
– several frontends for `C`, `Ada`, `Scade`, …
– a translation tool chain from the high level language towards a bit level language,
– a proof engine working over the bit level language,
– a proof verification engine,
– several tools for animation and debugging.

The performance of the proof engine allows us and our clients to manage the proof of industrial size problems some of them we will mention in the next section. The size of those models routinely topped ten millions variables and several hundred millions of clauses.

The qualification of S3 is made possible by the use of several diversified tool-chain with some small key tools built with respect to the higher integrity constraints that may be required — namely, an equivalence model constructor, and a tool to verify the validity of the proof (see fig. 1 on the following page).

[4] S3 is maintained, developed and distributed by Systerel.

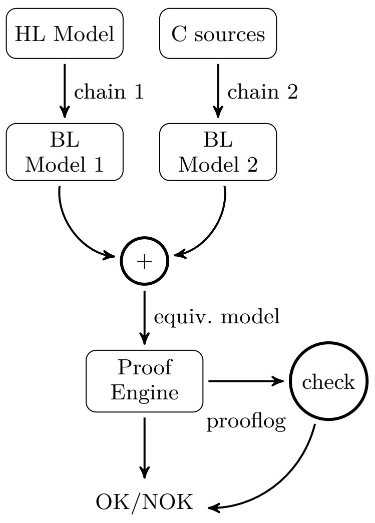


Fig. 1. Example of application for proving equivalence between C sources and a HL model specification. The two tool-chains are diversified and the software in the circles are developed with the higher level of integrity (HL = High Level, BL = Bit Level).

2.2 Floating-Point Arithmetic Library in S3

Critical applications used to use fixed-point arithmetic that requires less memory and less processor time then floating-point arithmetic (`FPA`) to perform non-integer computations on executing processors with no Floating-Point Unit (`FPU`), while leading to a limited-precision. Floating-point numbers support a trade-off between range and precision due to its formulaic representation which approximates a real number. Another advantage stands in the existence of a standard [1] based on solid mathematical grounds. Nowadays, `FPA` is more and more used in the space, aeronautics and automotive industries, due to the increasing complexity of the computations and because `FPUs` are becoming standard for most processors. Floating-point numbers are not real numbers. Floating-point operations behave in quite different way from the real counterparts, due, for instance, to rounding and cancellations [17]. Consequently, a software implementation of some mathematical expression usually provides results that are not strictly, mathematically, exact. As it is often difficult to foresee the behavior of floating-point programs, formal verification of floating-point programs is highly desired in the industry.

The basic approaches to address formal verification of floating-point programs include abstract interpretation, formal proof and bit-blasting (also called bit-flattening). Abstract interpretation partially executes program on an abstract domain. This approach performs well in program analysis with floating-point variables [2]. Formal proof supported by proof assistants is a very powerful approach that requires to be guided by highly skilled expert to direct the reasoning towards target properties. Interactive theorem provers such as `ACL2`, `Coq`, `HOL Light` and `PVS` have been applied to floating-point verification [14]. Both of abstract interpretation and formal proof approaches lack ability to generate counterexamples when the property does not hold. Bit-blasting represents floating-point operations as circuits, which are then transformed to Boolean formula with bit-wise operators to be solved by `SAT` solvers. Bit-blasting relying on `SAT` solvers is a fully automatic reasoning benefiting from counterexamples for floating-point programs. It is implemented in the Satisfiability Modulo Theories (`SMT`) solvers such as `Z3` [10], `MathSAT 5` [7], `SONOLAR` [15] and `CBMC` [5]. It is also the elementary but the most significant part of other floating-point verification strategies using `SAT` solvers as the back-end. Since the publication of `SMT-LIB` theory of binary `FPA` [18], solvers are starting to support it using some advanced `QF_FP` solving strategies, such as mixed abstraction in `CBMC` [5], non-conservation approximations in `Z3` [13], abstraction into interval arithmetic in `MathSAT` [3,4], translation into non-linear reals in `Realizer` [16], etc.

`S3` will be used to verify part of the floating-point software embedded in a rover platform, `TwIRTee` (a three-wheeled autonomous rover) used as the demonstrator for the `INGEQUIP` project. The `INGEQUIP` research project at the *Institut de Recherche Technologique (IRT) Saint-Exupéry* in Toulouse addresses the following equipment engineering topics: architecture modelling and evaluation, early `V&V` using virtual platforms, and formal verification. For this purpose, we implemented an `FPA` standalone library to enable the floating-point verification by means of bit-blasting. This optimized `FPA` library will establish a solid foundation and basic strategy for our future investigation on advanced `FPA` strategies in `S3`.

The implementation of `FPA` library in `S3` is based on the `IEEE FPA` standard 754-2008 [1]. The `FPA` library in `S3` includes:

– Binary interchange format encoding that allows user-defined ranges of exponent and mantissa, including single and double precisions
– Normal, subnormal, infinity, NaN numbers
– 5 rounding directions: `roundTiesToEven`, `roundTiesToAway`, `roundTowardPositive`, `roundTowardNegative`, `roundTowardZero`
– Comparison operations: `Equal`, `NotEqual`, `Greater`, `GreaterEqual`, `Less` and `LessEqual`
– Arithmetic operations: `Addition`, `Subtraction`, `Multiplication`, `Division` and `SquareRoot`
– Conversion operations: `convertIntToFloat` and `convertFloatToInt`
– Trigonometric operations: `Sin`, `Cos`, and `Tan`
– Default exception handling: invalid operations, division by zero, overflow and underflow

The *SquareRoot* and trigonometric operations are implemented by means of both interpolation table and the function proposed by Cody and Waite [8].

3 Industrial Applications

3.1 Railway Use Case — Interlocking Safety Proof

An interlocking is an arrangement of signal apparatuses that prevents conflicting movements through an arrangement of tracks such as junctions or crossings. An interlocking is designed so that it is impossible to display a signal to proceed unless the route to be used is proven safe.

The main challenge is to ensure that whatever scenario will happen, the designed interlocking will stay safe. And albeit its apparent simplicity one quickly understands the real underlying complexity — whatever rule you may imagine, it looks like one can always find a scenario breaking it (train can go backwards, can sometime appear to *fly* due to concurrency artefacts, ...)

Instanciation Design Process In a recent work, we have considered a Computer Based Interlocking (CBI) designed through an instantiation process. In such a process, the *signaling principles* are captured by the engineers to produce the Generic Design in some suitable language. This design contains a set of generic descriptions of code parts that an interlocking system shall execute for some specific object of the system such as signals, switches, routes,... Each of these generic descriptions is given in a parametrized form which allows for the specificities of the object to which it applies. For example, the generic design for a route will probably be parametrized with the list of switches contained in the route. The generic design is thus specific to a set of signaling principles, but independent of a particular track layout.

The generic design is then instantiated upon a particular track layout to produce the running software.

Verification Process We graft our verification process onto the instantiation scheme by designing a Generic Safety Specification made of:

– some high level proof obligations (3 to 4 properties about absence of derailments and collisions),
– some intermediate predicates modeling the domain (such as topological predicates which encode the track layout, integration predicates which associate input and output variables with their object instance, helper predicates conveying high level relations between objects such as reachability,...)
– an environment model upon which the proof obligations are expressed (trains behavior,...).

and then instantiate those properties and models with the specific track layout data. The overall uncertified process is given in figure 2.

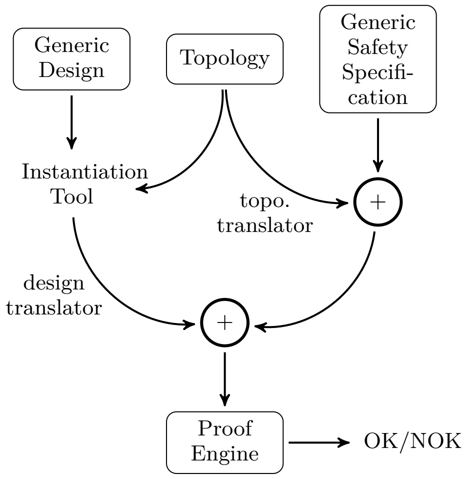


Fig. 2. Overview of an uncertified verification process for an *instantiated design*. The specific *topology translator* tool translates topological data in an High Level Language model. The specific *design translator* tool translates the instantiated design in a High Level Language model. The concatenation of all models is then feed to S3 proof engine. Manual lemmas may be added into the Generic Safety Specification to help the proof.

As an example of intermediate predicates used in the Generic Safety Specification, the fact that "every block belonging to a given route" is free can be expressed as:

```
ALL b:BLOCKS (contains(route,b) -> free(b))
```

where `ALL` is the universal quantifier, `b:BLOCKS` indicates that `b` shall range on every instances of the `BLOCKS`, and `->` is the implication operator.

Results The proof process was applied to a CBI of more than 200 routes and permit us to pinpoint 3 safety counterexamples which where then corrected whether in the design or in the topology data. The overall proof took less than 25 mn in the worst case, which would go as high as 1 hour for a certified process implying the use of a second tool chain, an equivalence verification and a prooflog check.

The verification was successful and proved to be usable enough to assist the design team in debugging the CBI. It also paved the way for a certified verification that would be used by the safety team.

In the competitive interlocking market, this would clearly give an edge to the CBI manufacturer when comparing to other solutions integrating or not formal verification.

3.2 Aerospace and Automotive Use Case

The floating-point verification by S3 has been applied to two case studies: the avionic triplex sensor voter and the automatic rover protection system. It is also used to automatically generate the test cases.

Triplex Sensor Voter Case Study The triplex sensor voter[5] is used in a common form of redundant aircraft system Triplex Modular Redundancy (`TMR`), which relies on three identical sensors to compute an output value from the three input values by the voter. It is implemented using linear arithmetic operations as well as conditional expressions (such as saturation). Its formal analysis covers functional and non-functional properties including stability, absence of runtime errors, and also to parameterize certain parts of the model to help the formal analysis. The formal analysis of triplex sensor voter was first studied by Dajani-Brown et al. in [9], where real values were abstracted by integer values and integrators were not used. In [11], Dierkes analyzed the `Simulink` model with real numbers by both simulation and formal verification, then estimated the impact of rounding errors caused by the floating-point implementation using `SMT` solvers and abstract interpretation. In [6], Champion et al. strengthened the stability property by generating lemmas using a property-directed approach.

[5] Triplex sensor voter case study is provided by Rockwell Collins to make it publicly available to the research community.

In our work, we start from a SCADE model of the voter and translate it to HLL model using the SCADE-translator. Thanks to our FPA library, the HLL model is then verified using the S3 solver. In parallel, we use SMT-solvers Z3 v4.4, MathSAT 5 and SONOLAR to verify the SMT model. Experiments are carried out using both simple and double precision floating-point numbers with or without subnormal numbers and with different rounding modes.

The results show that neither Z3 v4.4 (bit-blasting strategy, floating-point strategy) nor MathSAT 5 (bit-blasting strategy, abstract CDCL algorithm) or SONOLAR are able to handle the step instance, be it in simple or double precision. We managed to prove the inductive instance using a combination of SONOLAR bit-blasting to a CNF and a pure SAT solver (glucose 4.0 multithread with 8 threads and aggressive restart strategies, satellite preprocessing) in 10min of computation for the simple precision instance, and 4h15min of computation for the double precision instance (wall clock time). S3 proved the step instance in 6min using glucose 4.0 and 5mins using S3's own solver for the simple precision instance, and in 9h32min using glucose 4.0 for the double precision instance.

Test Generation In order to conform to the domain standards, companies sometimes have to setup processes where a structural test coverage must be achieved by mean of manual testing activities. Those activities consist mainly in finding the inputs and oracles such that the structural coverage criterion is respected.

We successfully used *model checking* with S3 in order to setup an automatic process of finding the set of tests that will achieve the structural node coverage on a Scade model, where the criterion was that at most one entry of a given node should change at once. Automation of such a task leads to great *cost reductions*.

Formal Verification for ARP Case Study The Automatic Rover Protection (ARP) system, a simple collision avoidance function, is developed for the twIRTee platform in the INGEQUIP project. It performs a predefined trajectory (a "mission") on a predefined track and avoid collisions with other rovers. The set of tracks are statically defined and embedded in the rover. The ARP is based on the following principles:

1. Missions (trajectories) are precomputed using an adapted shortest path algorithm[12].
2. A rover shall only move on a reserved path if there is a free reserved space ahead.
3. A reserved path is a stack of reserved nodes.
4. A rover resends reservation request to get the exclusive access to the tracks located ahead of it, if there is not enough reserved space (D_REQ) ahead.
5. A rover stops if there is not enough reserved space (D_STOP) ahead.
6. A rover reserves enough space (D_RSV) ahead for each request.
7. In order to ensure a consistent management of node reservations, a distribution strategy using id-priority is implemented.

In this case, S3 is exploited to verify several properties applicable to all independent rovers, such as $\mathbf{P}_1$: *rovers are never granted simultaneous access to the same area* (safety), and $\mathbf{P}_2$: *all rovers eventually reach their destinations* (bounded liveness), etc.

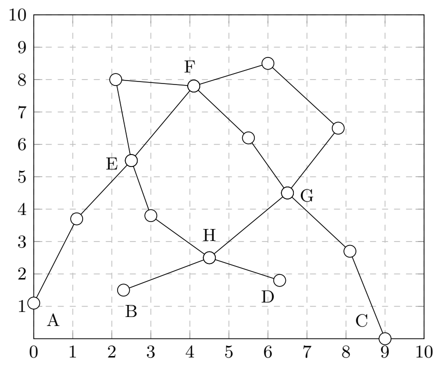


Fig. 3. Rover Trajectories for ARP

We present experimental results from a ARP case study with two rovers. The trajectories of rovers within an enclosed space of 10m×10m are predefined (see fig. 3). The rover R_1 proceeds with a speed $\mathrm{v}_1 = 0.4\mathrm{m/s}$ from node A, through node E, F, G and H, then stops at node

B, while the rover R_2 proceeds with a speed $v_2 = 0.3$m/s from node C, through nodes G, F, E and H, then stops at node D. The length of each trajectory is about 20m. As the first step of our study, we assume that the space occupied by rovers are ignored, and that the clocks of rovers are synchronized. Consequently, rovers are considered to perform actions synchronously. We proved the property $\mathbf{P}_1$ in almost no time using induction and the property $\mathbf{P}_2$ in several seconds using bounded model checking in S3.

3.3 Other Use Case — Constraint Solving for Matrix Wiring

S3 was used as a constraint solver to automate finding of solution to a wiring problem over a 100×100 matrix with several layers of wires and several limitations over the deformations that wires may undergo.

Once the counterexample obtained, a small and easy tool allowed to translate it into a *netlist*.

4 Conclusion and Future Work

Our experience with *model checking* has shown that S3 is a great tool to manage different types of industrial size problems with respect to their specific regulatory constraints. However, the tool is still at pains for some peculiar problems and has some intrinsic limitations that prevent its application on a wider range of problems, one of it being its current inability to handle designs which make use of floating-point arithmetic.

In order to tackle the floating-point limitation, we have designed an FPA library and integrated it in S3. Our next goal is to investigate how well it works on a wider range of industrial designs from automotive, aeronautic, industry, energy… and to establish benchmark (or reuse existing SMT benchmarks) to evaluate the performance of floating-point verification by S3.

The proof engine at the core of S3 is also undergoing heavy work in order to improve its efficiency for big size models.

This significant improvement, associated to the heavy work ongoing on the proof engine at the core of S3, shows great promises for dealing with future industrial challenges.

References

1. IEEE Standards Association. Ieee standard for floating-point arithmetic. 2008.
2. Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérome Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *ACM SIGPLAN Notices*, volume 38, pages 196–207. ACM, 2003.
3. Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening. Interpolation-based verification of floating-point programs with abstract cdcl. In *Static Analysis*, pages 412–432. Springer, 2013.
4. Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening. Deciding floating-point logic with abstract conflict driven clause learning. *Formal Methods in System Design*, 45(2):213–245, 2014.
5. Angelo Brillout, Daniel Kroening, and Thomas Wahl. Mixed abstractions for floating-point arithmetic. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 69–76. IEEE, 2009.
6. Adrien Champion, Rémi Delmas, and Michael Dierkes. Generating property-directed potential invariants by quantifier elimination in a k-induction-based framework. *Science of Computer Programming*, 103:71–87, 2015.
7. Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107. Springer, 2013.
8. W.J. Cody and W.M.C. Waite. *Software manual for the elementary functions*. Prentice-Hall series in computational mathematics. Prentice-Hall, 1980.
9. Samar Dajani-Brown, Darren Cofer, Gary Hartmann, and Steve Pratt. Formal modeling and analysis of an avionics triplex sensor voter. In *Model Checking Software*, pages 34–48. Springer, 2003.
10. Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
11. Michael Dierkes. Formal analysis of a triplex sensor voter in an industrial context. In *Formal*

Methods for Industrial Critical Systems, pages 102–116. Springer, 2011.

12. Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
13. Andreas Fröhlich, Armin Biere, Christoph M Wintersteiger, and Youssef Hamadi. Stochastic local search for satisfiability modulo theories. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
14. John Harrison. Floating-point verification using theorem proving. In *Formal Methods for Hardware Verification*, pages 211–242. Springer, 2006.
15. F Lapschies, J Peleska, E Gorbachuk, and T Mangels. sonolar smt-solver. *Satisfiability modulo theories competition; system description*, 2012.
16. Miriam Leeser, Sayan Mukherjee, Jaideep Ramachandran, and Thomas Wahl. Make it real: Effective floating-point reasoning via exact arithmetic. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4. IEEE, 2014.
17. David Monniaux. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(3):12, 2008.
18. Philipp Rümmer and Thomas Wahl. An smt-lib theory of binary floating-point arithmetic. In *International Workshop on Satisfiability Modulo Theories (SMT)*, 2010.

Session 4

Applicative domain

Wednesday 27th, 11:00 – Ariane 2

1 IoT Safety&Security challenges

Internet of Things (IoT) is a buzzword for the term “connected devices”, which we used in the last 15 years to describe the trend of connecting embedded devices. It is driven by the need to shift the usage of embedded devices to the next level of efficiency. By enabling seamless communication between all embedded devices, we capture more data about the ongoing process and are able to influence the process for optimization. The realization of seamless connectivity is favored by the broad availability of Ethernet and wireless communication. Even real-time requirements are covered by specific industrial Ethernet protocols, which are fast enough to drive high frequency control systems with a small amount of jitter.

With this seamless communication, functional safety and security of an embedded/IoT device become a challenge. A communication channel is always subject for vulnerability and can compromise the safety of the system because of a modified configuration. A security attack to a water heating system, which provides boiling water instead of tempered water, or a control valve for a chemical process, which is ‘out of control’ will definitely evolve to a safety problem.

The functional safety of a system ensures, that a system does not harm its environment. Safety is ensured by implementing a safety concept in software and/or hardware. On the software side, the underlying software platform (Operating System and Middleware) should support safety in its design and architecture, so that functional safety requirement can be more easily implemented.

Security concepts shall ensure, that the system is not harmed/attacked by its environment. As security problems can evolve to serious safety problems, security can also be seen as an indispensable safety concept. One aspect of security is to keep data confidential and ensure its integrity. If data confidentiality and integrity are not secured by appropriate strategies, a medical device may harm the patient with the wrong dosage. If data or the system is not available because of a Denial of Service (DoS) attack to the medical system, the medical device may not function at all.

With these scenarios in mind, we need to look for a design methodology, which bears safety and security in its architecture. In the functional safety domain, the use of a separation kernel is a well known solution for safety and security by design concept. In a nutshell, a separation kernel allows spatial and temporal separation between applications, by providing **separated partitions** for application execution and a concept for **sharing CPU time**. A thoroughly separated application concept by using a separation kernel guarantees non-interference, so that errors cannot propagate from one application to another.

From a safety perspective this partitioning mechanism is used to isolate applications of different criticality so that risk reduction can be applied for each application/partition independently. The application isolation capabilities are also suitable for system security by isolating critical data from non-critical data and allowing only controlled information flow between isolated partitions. Controlled information flow can be realized by:

- A white list policy for inter-partition-communication. That is, communication is generally prohibited and needs to be allowed explicitly, defined at system configuration, not at runtime.
- Using cryptography before exchanging data between partitions and with the outside world
- Applying a security policy for data communication so that communication is monitored and controlled

This paper uses the PikeOS Separation Kernel as an example implementation and explains how safety and security aspects are covered by the separation kernel architecture. An automotive use-case designed by Continental and an IoT gateway design based in the Freescale IoT gateway will be introduced by describing the chosen safety and security concept.

2 Separation Kernel - Safe and Secure Real-Time Virtualization

A well-known approach for separation is an Hypervisor, which is used in the IT domain to run an operating system on top of another operating system. Most Popular implementation like VirtualBox, VmWare etc. provide an execution environment to run a COTS operating system like Linux on another COTS OS. As the afore-mentioned technologies were never architected with safety and security in mind, this separation does not allow enough safety and security for certification.

A separation kernel like PikeOS follows a slightly different approach. It uses a microkernel as the underlying operating system architecture and provides means for partitioning on top of the microkernel. The partitions can be used as an execution environment for:

- Bare metal applications
- Applications using the microkernel API, PikeOS Native.
- Using a standard API like POSIX.
- Using any run-time environment like Java, Ada including Ravenscar profile.
- Running an operating system like Linux, AUTOSAR, ARINC-653 in a partition …

The main advantage of this approach is, that the microkernel provides real-time capabilities and separation capabilities in the operating system design. Thus we have a Real-time Operating system and a hypervisor architected into one product.

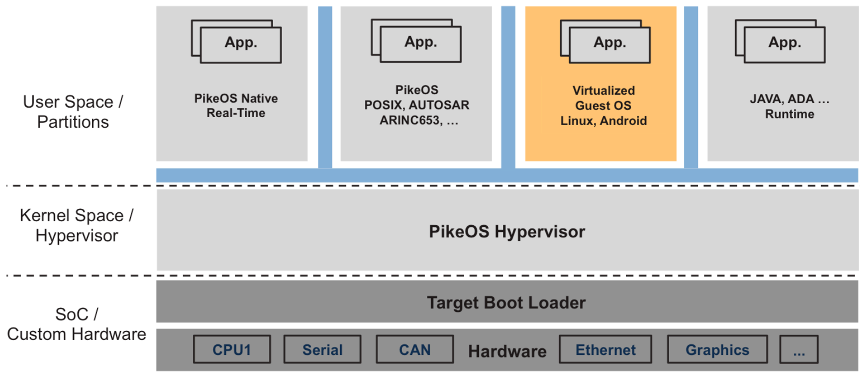


Figure 1: Separation Kernel architecture

The architectural concept of a separation kernel is based on separating resources, so that applications cannot interfere with each other. The available resources on computing hardware are the physical hardware components and the CPU execution time. The separation of physical resources is called spatial separation or resource partitioning, while the separation of the available execution time is known as temporal separation or time partitioning.

2.1.1 Divide-Et-Impera – Resource Partitioning

Spatial separation is achieved by means of resource partitioning, in which system resources such as memory, files, devices, secure communication channels and cores are statically assigned to different resource partitions. All applications run in the context of a resource partition and PikeOS ensures that during runtime an application has guaranteed access to the assigned resources of its partition and these resources are not accessible from applications belonging to other partitions.

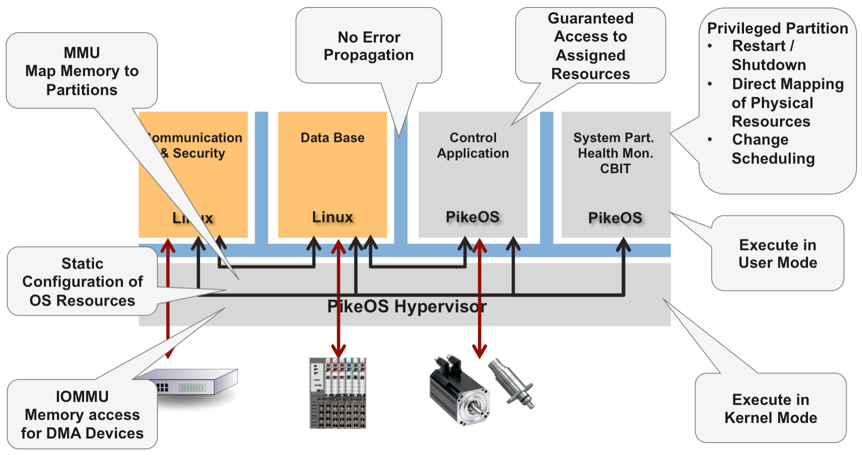


Figure 2: Resource Partitioning

Resource partitioning is enforced by using the MMU to control access to the available resources. Each hardware device is somehow represented by a physical address in order to access this device. Resource partitioning is realized by using the MMU to map a certain memory are into a partitions virtual memory space and allow or deny access to this RAM and I/O devices. The configuration of the MMU is done statically at system startup in the PikeOS microkernel and it is **not** modifiable at run-time, so that a hacker cannot modify the resource configuration later on.

DMA capable devices can bypass the MMU protection and access memory areas, which are reserved for the operating system or an application partition. To control the memory access of DMA bus master devices, the SoC provides an additional MMU device, which maps device-visible virtual addresses to physical addresses. The processors vendors name this mechanism differently so that Freescale (PowerPC) is talking about a Platform Management Unit (PAMU), ARM processors name this a System Memory Management Unit (SMMU) and Intel calls this VT-D on x86 architecture.

Last but not least the privilege level of the processor is used to configure partitions to run in user-space and the operating system to run in kernel-space. As user-space applications have no direct access to kernel-space, the operating system is safe against any misbehavior of user-space applications. In case of a separation kernel, partitions run in user-space (see Figure 1).

In summary, the separation makes sure that:

- A partition has guaranteed access to devices, which are configured only for the partition
- A partition has no access to a device, if it has not been configured for accessibility
- That errors cannot propagate from one partition to another partition

2.1.2 Time-Partitioning

Temporal separation is provided by means of Time Partitioning in which the CPU time is divided into time windows. The duration and order of these time windows are statically defined and enforced by the PikeOS Microkernel during runtime. Applications are assigned to one or more of such time windows and are considered for scheduling only when the associated time window is active. In this way the temporal behavior of a partitioned application is made independent from the rest of the system.

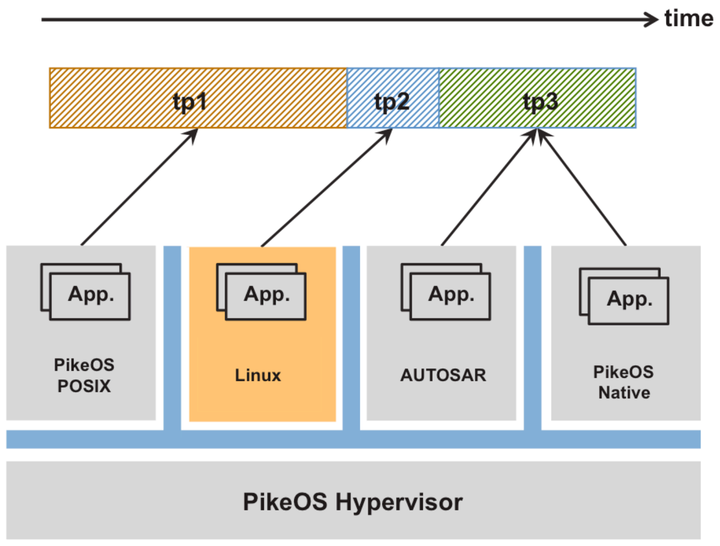


Figure 3: Time Partitioning

Priority based scheduling and a preemptive kernel design provides real-time capabilities to a microkernel, but this does not always guarantee a calculable deterministic behavior of the system. The more complex a system configuration is (multi-core is one of the major complexity issues), the more difficult it will be to determine a worst case execution time (WCET) for an application running in this configuration. The WCET analysis is evident for the safety as it is the guaranteed maximum time for the system's response. Time partitioning demerges the complexity of the run-time behavior of a system by assigning a partition's applications to a dedicated time window. If this time-window is active, only the defined subset of applications is executed.

As pure time partition would not allow the handling of exceptions or the reuse of non-consumed time, adaptive time partitioning provides the required flexibility. The patented time-partition zero (tp0) concept of SYSGO defines an additional time partition (tp0), which is always overlapped to the current eligible time-partition. If an executable (thread) in tp0 has a higher priority than threads in the current active time-partition, than the high-priority thread in tp0 will be eligible to run. The opposite example is, if there is nothing to do in the active time partition, threads from tp0 are recognized for execution, thus allowing low priority threads assigned to tp0 to use not-used execution time.

Complex scenarios with multiple cores and multiple applications can have a complexity, which makes it impossible to calculate a WCET with the available means. The time partition based approach reduces the complexity of a system, so that the WCET is able to be calculated and stays is a reasonable range.

2.2 Separation Kernel Safety

An Application, which runs in a partitioned system, is not safe per se. The application safety is defined by a set of safety requirements, which need to be implemented in software and/or hardware. The important aspect of the separation Kernel is the possibility to design safety-critical software with the means provided by the separation Kernel. The major safety concept is the separation of application and the controlled information flow between the separated entities.

Industrial safety standards require the certification of software, which means that application code must be thoroughly tested and documented. These standards define several levels of criticality, which are measured by the severity of the hazards caused by application failure. In case of EN-50128 and IEC-61508 the criticality is ranked from Safety Integrity Level (SIL) 0 being the most basic up to SIL-4 being the most stringent level. The avionic safety standard DO-178B/C defines Design assurance levels ranked from E (basic) to A (strict). As separation kernels allow noninterfering application partitioning, the system can be designed with mixed criticality levels for each partition. That is, each partition can host an application, whose criticality to the system safety is different. This concept is market proven and has undergone several certifications according to industrial standards like IEC61508, EN50128, DO-178B and ISO26262.

Safety certification standards define strict guidelines for the development and deployment life cycle in order to achieve a conformity certificate. Beside the strict processes, extensive testing is required for each line of code. Test code and testing effort grows with the criticality level of the application. As the costs grow drastically with each additional line of code, there is a clear advantage to separated critical from non-critical application and keep the “Trusted Code Base” (TCB) for critical application components as small a possible.

Another important aspect for safety is the above-discussed WCET and thus the real-time behavior of the underlying operating system. Safety critical software requires a detailed timing analysis in order to prove that the software can react accurately in critical situations. The time partition concept demerges the complexity of the runtime behavior and eases the timing analysis significantly.

2.3 Separation Kernel Security

While safety is the attempt to protect the user from harm, resulting from malicious behavior of the system, security aims to protect the systems against malicious attacks from the humans, interfacing with the system. Security architects have the following aspect in mind, when designing a system:

- **Integrity:** Make sure, that the data cannot be altered without authorization
- **Confidentiality:** Make sure, that the data cannot be accesses by anybody who is not authorized to access the data
- **Authentication:** Prove the identity of a person who requests access to the system
- **Access control:** regulate the access to the system
- **Availability of resources:** Make sure that the system is available and is not hindered in operation
- **Non-Repudiation:** Have a proven concept so that nobody can deny his access to a device later on

In general-purpose platforms, these concepts are realized by using state of the art security technology like

- Firewall technology that monitors and controls the incoming and outgoing network traffic
- Authentication mechanisms to make sure that only authorized personal has access to the device
- Cryptography to encrypt data before transferring it over vulnerable transmission channels
- Monitoring and perimeter protection in order to identify and actively protect against intrusion

The usage of a separation kernel enriches these available security concepts with a fundamental security architecture, which we name as “security by separation and controlled information flow”.

“Security by separation” means, that sensitive data can be isolated into a partition and “Security by controlled information flow” means that access to the isolated sensitive data is controlled by the configuration of the communication means used for inter-partition-communication. The most secure approach is not to allow any communication, but this is not in the intention of the inventor. Communication from and into a partition with sensitive data can be secured by using/implementing a security policy manager, which captures all communication data and applies a security check (based on cryptography algorithm if required) on the origin and the content of the data.

Security certification standards such as IEC-15408 (Common Criteria) or the IEC-62443 specify the security functional and assurance requirements which have to be fulfilled by the device and thus by the underlying software platform. The IEC-15408 describes the security requirements for general-purpose equipment and the IEC-62443 targets Industrial Automation and Control Systems (IACS).

Common Criteria (IEC-15408) provides assurance that the process of specification, implementation and evaluation of a computer system (IoT devices are computer systems) has been conducted in accordance to the standard. An Evaluation process validates the security claims, which have been made on the device, so that the standard defines several Evaluation Assurance Levels (EAL), beginning from 1 being the most basic up to 7 being the most stringent level describing the depth and rigor of an evaluation.

To have a more generic certification, SYSGO has an ongoing project to certify PikeOS to CC EAL5+. It is important to know, that general purpose computing platforms (like Windows and Linux) have achieved EAL4 so far, but separation kernels like PikeOS can achieve EAL6.

2.4 Safe&Secure Automotive application

The high level of connectivity in a car makes it a perfect example of an IoT device. The Security of a car was heavily discussed in the press because a couple of automotive solutions were hacked in the past month. Specifically the hack of the Jeep Cherokee demonstrated, that a security leak could cause severe safety problems. Videos on the web show, that a hacker is able to steer the car and even operate the breaks.

The implementation shown in Figure 4 is a combination of a cockpit showing the cars speed and rotation speed. Motor status information is received over the CAN bus and displayed on a dedicated display which is integrated in the cockpit. Speed and rotation speed indicator reside in a dedicated partition. The car status is calculated and controlled in another partition.

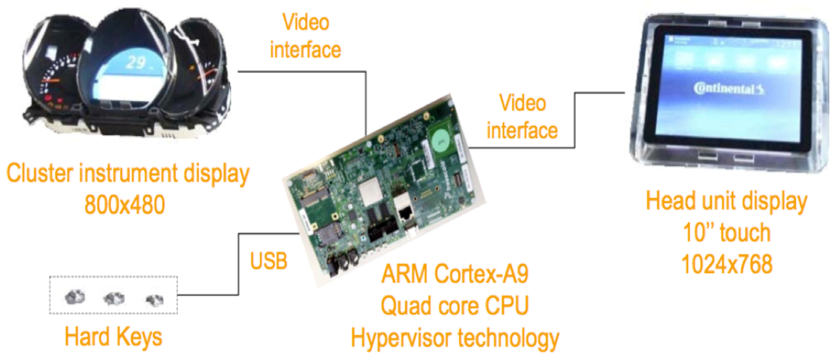

Video interface
Video interface
Cluster instrument display 800x480
USB
Hard Keys
ARM Cortex-A9 Quad core CPU Hypervisor technology
Head unit display 10" touch 1024x768

Figure 4: Continental Integrated Cockpit and Infotainment design based on PikeOS

The head unit display shows an Android, which runs in a dedicated PikeOS partition. Android is used to provide infotainment and navigation functionality. Browsing the Internet and downloading and installing Android apps offers the possibility to configure the head-unit according to the end-users preferences.

Android is the interface to the outside world and can be subject to attacks from the outside world. If a hacker is able to highjack the Android head-unit over a known vulnerability, than even PikeOS cannot prevent the access to the Android system. Starting from the Android partition the hacker will try to access physical devices in order to change system behavior. Or he may try to modify the operating system configuration in order to get control over the entire cockpit. Or, he may try to access other partitions (speed or car status) in order to modify functionality. Our Separation Kernel will deny all tries from the hacker to access I/O devices if they have not been assigned to the Android partition. Accessing the operating system from the Android partition will not be possible because the Android partition operates in user space and does not have the privileges to access the OS.

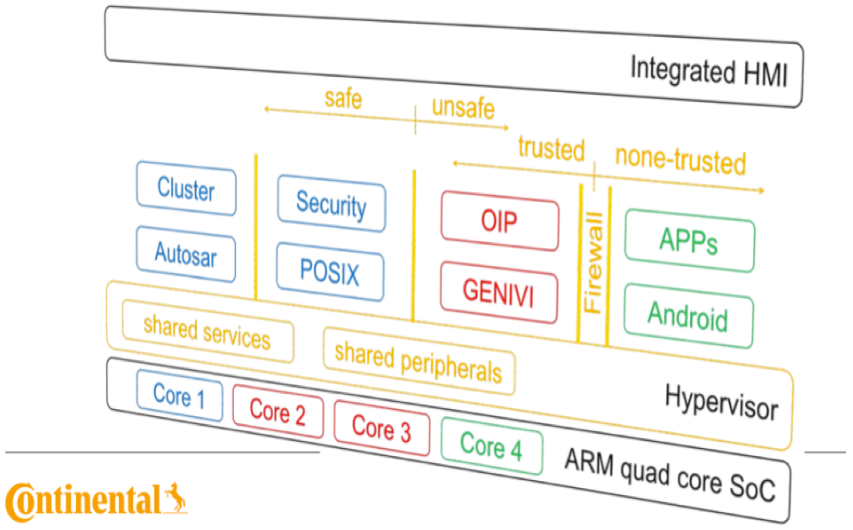


Figure 5: Integrated Cockpit and Infotainment architecture

2.5 Safe&Secure IoT Gateway

The Freescale Layerscape 1021A (further named as LS-1) is an ARM based SoC (see Figure 6). The I/O offered by the LS-1 is quite suitable for SOHO gateways. Freescale provides an IoT gateway reference design, which is supported with a PikeOS board support package (BSP) so that partitioning based safety and security can be implemented. The LS-1 uses the Freescale QorIQ framework so that secure-boot and secure-update can be implemented by using the Trust Architecture.

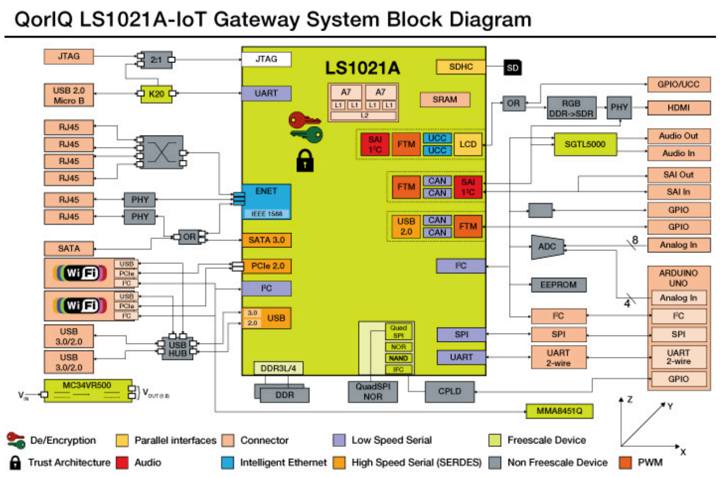


Figure 6: Freescale IoT Gateway Block Diagram

The implementation shown in Figure 7 was derived from the idea of a customer to offer a gateway, which offers the end-users the flexibility to customize their Linux domain with any open-source software downloaded from the Internet and provide home automation functionality. The service provider, who sells/leases the gateway to the end-user owns a dedicated Linux domain which allows him to offer services to the end-user by uploading the service software to his service-provider domain. The data of each Linux partition is separated by using a dedicated network, hosted by dedicated PikeOS partitions. The storage is managed by a PikeOS volume provider/manager. The volume manager provides a dedicated volume to each Linux domain with configured access rights. The two Ethernet interfaces are separated so that network problems do not propagate into the other partitions.

The final implementation offers direct IO access from the Linux-1 domain to the underlying hardware. Thus additional home-automation functionality can be implemented in the user-domain.

The shown architecture offers a certain level of security by isolating the data of the user domain and the service-provider domain so that a user cannot access the service provider's configuration and the service provider shall not access private user data. At a first glance, safety is not a demand for this kind of devices, but recognizing the fact that the gateway does also provide home automation functionality, the separation based security will ensure the safe operation of the home automation. All partitions can be updated independently, so that service and user applications can be updated on the fly without shutting down the entire system.

The idea to extend a gateway or router with home automation functionality in not new and major vendors in this market (like AVM, D-Link, etc.) have adopted this idea with their new generation of products. Their security concept is mainly based on using Linux security means and applying security updates. This works quite well if security updates are available in time for the kernel version. If a hacker is able to highjack the Linux system over an unpatched vulnerability, the hacker will try to access physical devices in order to change system behavior. Or he may try to modify the operating system configuration in order to get control over the entire system. Or, he may try to access other application threads in order to steal data or modify functionality. **But**, as we have seen in chapter 2.4, the separation kernel will deny all tries from the hacker to access I/O devices and the operating system. Stealing data from another partition can be prohibited by securing the inter partition communication with a policy manager, which captures all communication data and applies a security check (based on cryptography algorithm if required) on the origin and the content of the data.

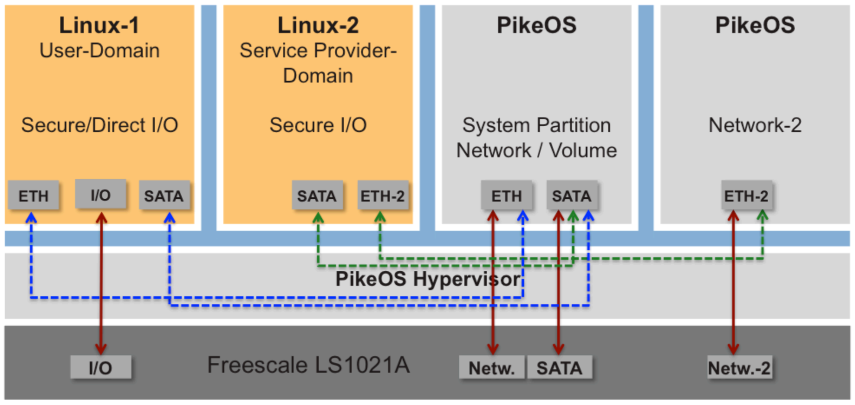


Figure 7: Separation Kernel Safety

3 Conclusion

IoT targets seamless communication between all devices from sensor level up to the enterprise level. These interconnection leads to a more vulnerable system design, which bears the risk to be a victim of a security attack. Specifically for IoT devices such as industrial control system, medical devices, transportation infrastructure, just to name a few, security attacks can compromise the systems safety by modifying the system configuration.

Security and safety is not a functionality, which you can just enable for a device. They need to be implemented following safety and security requirements and by sticking to the rules of industrial safety and security standards. Implementing safety and security becomes easier, if the underlying software platform provides an appropriate architecture for safe and secure design. An RTOS that is also an hypervisor based on a separation kernel like SYSGO PikeOS offers safety and security by separation and controlled information flow. Applications are isolated into partitions, which are protected against each other. The partitioning is non-interfering so that errors cannot propagate over partition boarders. The communication between partitions is configured and controlled so that access to safety and security critical data can be limited/prevented efficiently.

PikeOS has proven its safety and security concept by achieving a couple of safety and security certifications:

- IEC 61508 SIL-3
- EN-50128 SIL-4
- DO-178C DAL-A
- IEC 15408 EAL 5+ ongoing

Even more, PikeOS has achieved the worlds first EN 50128 SIL-4 certification on a multicore CPU.

A Distributed User-Centered Approach For Control in Ambient Robotic

N. Verstaevel[a, b] verstaev@irit.fr

C. Régis[b] regis@irit.fr

M.P. Gleizes[b] gleizes@irit.fr

F. Robert[a] fabrice.robert@sogeti.com

[a]Sogeti High Tech,
3 Chemin de Laporte, Toulouse, France

[b]IRIT, Équipe SMAC,
Université Paul Sabatier, Toulouse, France

Abstract: Designing a controller to supervise an ambient application is a complex task. Any change in the system composition or end-users needs involves re-performing the whole design process. Giving to each device the ability to self-adapt to both end-users and system dynamic is then an interesting challenge. This article contributes to this challenge by proposing an approach named Extreme Sensitive Robotic where the design is not guided by finality but by the functionalities provided. One functionality is then seen as an autonomous system, which can self-adapt to what it perceives from its environment (including human activity). We present ALEX, the first system built upon the Extreme Sensitive paradigm, a multi-agent system that learns to control one functionality in interaction with its environment from demonstrations performed by an end-user. We study through an evolutive experimentation how the combination of Extreme Sensitive Robotic paradigm and ALEX eases the maintenance and evolution of ambient systems. New sensors and effectors can be dynamically integrated in the system without requiring any action on the pre-existing components.

Keywords: *Distributed Architecture, Innovative Architecture, Human System Interactions, Control System, Internet of Things, Smart Devices, Adaptive Multi-Agent System*

1 INTRODUCTION

We are living at a time where technologies evolve every day. Intelligence, once restrained in personal computers, is now distributed in our environments under many forms. Those systems are *ambient* (Guivarch, 2012). Internet of things, wearable sensors, robotics, home automation, are illustrations of the ubiquitous computing revolution (Weiser, 1991). As software and hardware become ever more elaborated, intelligence is now embedded in objects. We have at our disposal libraries of various components realizing functions rather than objectives. For example, smart cameras can produce data from image recognition algorithms or every day object can play a role in the human-system interaction. Each of those components is autonomous and designed independently.

A robot for a particular application consists in the aggregation of the necessary components to satisfy its objectives. Those components could be part of the robot body or distributed in its environment. The collective of components has to interact and collaborate to perform an adequate global activity. The design of an intelligent system is then a matter of integration and a recurrent challenge is how to enable all those intelligent things to collaborate whereas they have an autonomous activity.

Those ambient systems are truly complex: a potentially huge number of heterogeneous devices evolves autonomously (including appearance or disappearance of devices) to provide services to its users (Perera, 2014). Designing an *ad hoc* controller supervising the whole activity involves having a lot of knowledge on the system dynamic. Any change in the system composition implies re-performing the whole design process meaning that sustainability of such system is a challenging task. Complexity is increased by the specific, multiple and often changing needs of end-users. Designers cannot make *a priori* a complete specification. Actually, the maintenance and evolution of an ambient system involves high costs, as it usually requires high knowledge and skills.

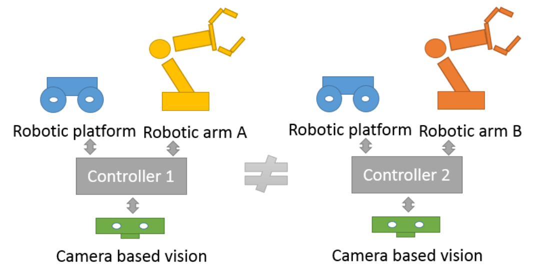


Figure 1: Example of integration problem: designing a unique controller for 3 robotic components is dependent of system composition.

One of the challenges is then to give to each device the ability to self-adapt to both system dynamic and end-user needs.

This paper contributes to this challenge by studying the benefits of using self-adaptive components in the design of robotic applications. The paper is organized as follows: First, we formulate the problem of integration of robotic. Secondly, we present the Extreme Sensitive Robotic paradigm, an innovative architecture to design ambient robotic applications. A scientific background is then provided to position the paper in regard with other scientific domains. Our main contribution, ALEX, is then presented in section 5. Section 6 proposes a use-case study of using ALEX in combination with the Extreme Sensitive Robotic paradigm from the viewpoint of a designer of a robotic application. Finally, we conclude with some perspectives.

2 DESIGNING A ROBOTIC SYSTEM: THE INTEGRATOR PROBLEM

The evolution of technologies, both in terms of hardware and software, makes available libraries of various components realizing functions rather than objectives. Internet of things (Perrera, 2014) is the perfect illustration of such a possibility. Designers of those systems have to aggregate different functions to build a system providing services to its users. Those functions are provided by electronic devices (basically by effectors). Each device exercises a control over a particular functionality. For example, a particular device can control the activation of an electric shutter and another one can control lights. Robots are part of those systems and propose a set of functionality, which can include mobility. A mobile robotic platform equipped with a robotic arm then provides two functionalities: the ability to move and the ability to grab objects. The integration of those different robotic components in order to provide service to humans is a complex task.

Let's consider the case of a designer who wants to integrate a robotic arm from one constructor and a mobile platform from another constructor, while using image processing algorithms from a third provider to perform a collecting task. The design of an *ad hoc* controller for such application is complex and requires a lot of expertise on each component, but also on its environment (which includes human activity). If for any reason, a component is replaced with another one (even if this new component provides the same functionality), the whole design process has to be performed again. This is time greedy and involves high cost of maintenance and evolution. However, the same system composition (one robotic arm, one camera based vision and a robotic platform) could be used in different kind of application. For example, one could want to use it for turning valves in a factory or the other for cleaning a room in a nuclear power plant. Each new application involves designing a new controller (figure 1).

A designer of such system would profits from a system capable of self-adapting to both the environment and users' needs without requiring reprogramming any system's component. This is the postulate made by *Extreme Sensitive Robotic*.

3 EXTREME SENSITIVE ROBOTIC: EXPECTATIONS

The Extreme Sensitive Robotic (XS Robotic) is an integrative approach of functions of perception, decision, action and interaction. It proposes a bottom-up approach focusing on functionality rather than a top-down approach focusing on objectives. Each function is an atomic part composing the micro-level of the system. A robot is then seen as the aggregation of the necessary functions to satisfy user's needs. Further, a group of robots or a whole ambient system has to be considered in the same way: a set of macro-functions (each robot) working in coordination.

The XS Robotic considers each robotic device in interaction with humans, other devices and the environment through sensors. Each device is autonomous which induces that the complexity of a robot (or a collective or robots) is not described explicitly or implicitly in it. Each device determines its own activity in interaction with its environment. The problem of integration then becomes a problem of adaptation. Each device has to adapt its behavior to its environment.

By applying the XS Robotic paradigm to the

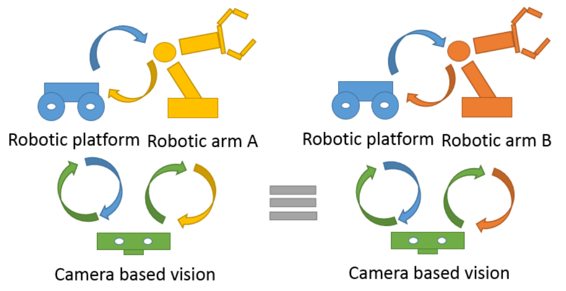


Figure 2: The same problem through the scope of XS Robotic. The two systems are an equivalent problem.

previously enounced problem, there is no difference between the two systems (Figure 2). Indeed, as each device is able to self-adapt to its environment, it will autonomously integrate any new device to its own activity.

To be truly effective, the XS Robotic needs generic algorithms allowing devices to self-adapt both to system’s dynamic and humans. Those devices, which interact with their environment and their users, must have the capacity to automatically learn from this interaction and exploit this knowledge. But to be as natural as possible, the human-system interaction must rest on a process that does not need any expert knowledge. On contrary, it must provide a natural way for any kind of user to express their needs.

4 SCIENTIFIC BACKGROUND

On the previous section, authors present the *Extreme Sensitive Robotic* as an integrative approach resting on self-adaptation skills. *XS Robotic* deals with the ability to learn from interaction with the environment and users. However, the idea of making autonomous systems able to self-adapt and learn from their environment is not completely new. In fact it relies in the heart of informatics. Yet in the early 50's, Alan Turing (Turing 1950) states that "instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain". On this section we present concepts coming from robotic, cognitive science and artificial intelligence that attempt to build autonomous artificial systems with the ability to learn from interaction. For each domain, we point out main properties required for enabling *XS Robotic*.

More than sixty years after the dream of Alan Turing, robotic controllers are still handcrafted. Artificial intelligence failed to bring Turing's dream to life. Brooks explains that this failure may come from engineer's conceptualization of the world that may not be appropriate for artificial systems with a different sensory motor apparatus (Brooks, 1990). Due to the limits of introspection, the abstraction that a human would supposed to be appropriate to build a system may be completely different to what he is actually using. A metaphor that sums up Brook's idea would be that making abstraction of the world is like *observing the world through a keyhole instead of opening the door*, depriving the system of all the wealth that this world has to offer. To avoid this problem, Brooks proposes the *physically grounding hypothesis* that stipulates that interaction with the environment has to be the primary source of constraint for the design of intelligent system.

Pfeifer (Pfeifer, 2006) goes further by arguing that there is a strong relationship between the body and the mind. Pfeifer states that the traditional view of intelligence is that it is located inside the brain, or more generally inside the control system. However, he shows that studying the brain (or the control) alone does not allow to completely infer the behavior of the system. The brain needs a body to act, and the way the brain is embodied in the physical world may strongly influence the way it acts. This relation is called *embodiment*. Embodiment plays an important role in learning as what we can learn is strongly related to what we can do.

Zlatev and Balkenius (Zlatev, 2001) state that cognitive science community realizes that “true intelligence in natural and (possibly) artificial systems presupposes three crucial properties:

- The **embodiment** of the system
- Its **situatedness** in a physical and social environment
- A prolonged **epigenetic developmental process** through which increasingly more complex cognitive structures emerge in the system as a result of interactions with the physical and social environment

Cognitive sciences have a particular echo inside the artificial intelligence community and has inspired learning techniques. (Guerin, 2011) proposed an overview of artificial intelligence approaches trying to build programs that could develop their own knowledge and abilities through interaction with the world. The approaches inventoried by Guerin share the same conception of the learning process. They see learning as an iterative process by which a system builds increasingly more complex structures, and uses these structures to behave in interaction with the environment. However, most of them fall under Brook's critics. Moreover, most of the methods only

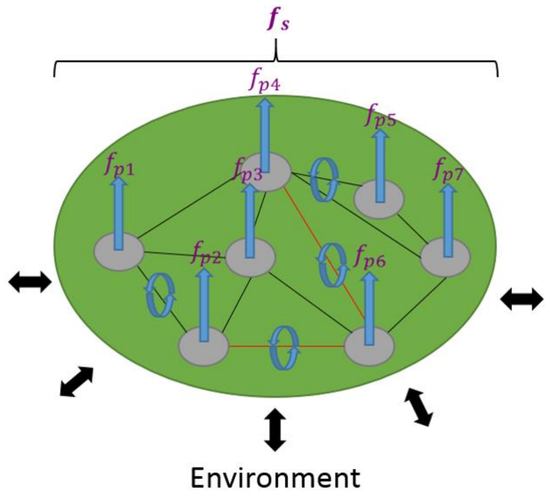


Figure 3: A schematic view of an AMAS system. The functionality f_s provided by the system is more than the sum of each agent functionality f_{p_i}. It is the result of interactions between agents and the environment.

took interest on knowledge creation, avoiding the problem of knowledge exploitation.

We agree with Brooks and Pfeifer vision of intelligence. That means that to be truly adaptive, the design of an *XS Function* should not fall into Brook's critic of abstraction. An XS *Function* must then exploit all source of information as a signal without making any abstraction on it. Semantic is then prohibited. On contrary, each signal has to be considered the same way, as a raw observation of the world.

Furthermore, as we cannot make a separation between the body and the mind, the learning process allowing self-adaptation has to be self-aware of its own activity and its consequences on what it senses from its environment. Learning from the consequences of my own embodiment relation (which means consequence of my own activity) will allow the system to sense any changes on this relation, either this changes come from a modification of system's body or environment. The learning process should be made through interaction with the physical and social environment by which a complex behavior emerges.

To be usable by any kind of user, the adaptation process must not require any expertise. *Learning from Demonstration* (Argall, 2009) appears then to be a promising approach. Learning from Demonstration is a paradigm to dynamically learn new behaviors from demonstrations performed by a human. The process of demonstration does not require expertise from the user on the controlled system while allowing the system to capture the user's needs.

On the next section, we present our contribution to enable the *XS Robotic* vision. This contribution is a combination of the *Adaptive Multi-Agent System approach* and *Learning from Demonstration*.

5 ADAPTIVE LEARNER BY EXPERIMENTS

Through the scope of XS Robotic, the problem of integration of robotic components is a problem of self-adaptation. We then need to propose an algorithm that enable each device to self-adapt. On this section, we present our contribution, ALEX, an adaptive multi-agent system designed to learn from demonstration performed by a tutor. Its design is based on the Adaptive Multi-Agent System (AMAS) approach.

5.1 AMAS approach

The *Adaptive Multi-Agent System* approach (Gleizes, 2012) addresses the problematic of complex systems with a bottom-up approach where the concept of cooperation is the core of self-organization. The theorem of functional adequacy (Camps, 1998) states that:

> "For all functionally adequate systems, there is at least one system with a cooperative internal state that realizes the same function in the same environment"

A general definition of *cooperation* could be the golden mean between altruism and selfishness (Picard, 2005). The role of an AMAS designer is to identify non cooperative situations and to propose mechanisms to anticipate or resolve such situations. The agent detecting a non-cooperative situation automatically triggers those mechanisms. Three mechanisms allow repairing or anticipating a non-cooperative situation (Capera, 2003):

- **Tuning**: the agent adjusts its internal state to modify its behavior,
- **Reorganization**: the agent modifies the way it interacts with its neighborhood,
- **Evolution**: the agent can create other agents or self-suppress when there is no other agent to produce a functionality or when a functionality is useless.

The system will then self-organize to stay in a cooperative state. From cooperative interactions between the system's entities emerges a global function that is more than the sum of the parts (Figure 3).

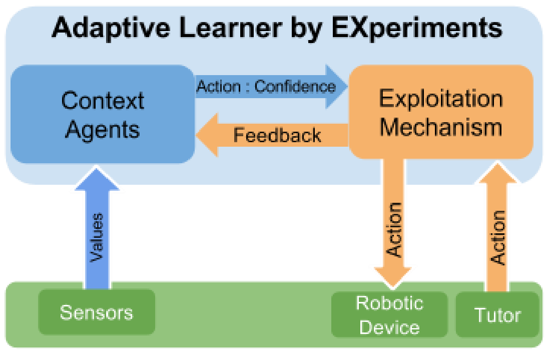


Figure 4: ALEX architecture

The approach proposes a methodology called ADELFE that guides the designer of an AMAS system (Bonjean, 2014).

5.2 Learning from Demonstrations

Learning from Demonstration, also named *Imitation Learning* or *Programming by Demonstration*, is a paradigm mainly studied in the robotic field that allows systems to self-discover new behaviors (Argall, 2009). It takes inspiration from the natural tendency of some animal species and humans to learn from the imitation of their congeners. The main idea is that an appropriate controller for a robotic device can be learnt from the observation of the performance of another entity (virtual or human) named as the ***tutor***. The tutor can interact with the system to explicit the desired behavior through the natural process of demonstration. A demonstration is then a set of successive actions performed by the tutor in a particular context. The learning system has to produce a mapping function correlating observations of the environment and tutor's actions to its own actions. The main advantage of such technique is that it needs no explicit programming or knowledge on the system. It only observes tutor's actions and current system context to learn a control policy and can be used by end-users without technical skills.

The paradigm has been used on a wide range of applications such as autonomous car following (Lefèvre, 2015), robot trajectory learning (Vukovic, 2015) or robot navigation in complex unstructured terrain (Silver, 2010). Recent surveys (Billard, 2008) (Argall, 2009) propose an overview of the LfD field illustrating a wide variety of applications. Our interest is not to focus on one particular application. On the contrary, we want to deal with any kind of ambient robotic system.

5.3 ALEX architecture and general behavior

In accordance with the ADELFE (Bonjean, 2014) methodology, we designed ALEX (Adaptive Learner by Experiment), an Adaptive Multi-Agent System, to learn to control a system from demonstrations.

On the rest of this section, we present ALEX architecture and focuses on Context agents, which are the core of the learning process.

5.3.1 ALEX architecture

An ALEX instance is designed to control a robotic device (an effector) by sending actions to it. Those actions are changes of the current state of the robotic device. An ALEX instance is in constant interaction with its environment from which it receives actions from its tutor and a set of sensors values. ALEX observes the current state of all accessible sensors, the action performed by the tutor and in response sends the action to be applied by the controlled robotic device. ALEX is composed of two components, an Exploitation mechanism and a set of Context agents. The figure 4 illustrates ALEX architecture.
The Exploitation mechanism is responsible for sending actions to the robotic device. In order to do so, it receives both the action performed by the tutor and a proposition of action from the set of Context agents. By comparing the action realized by the tutor to the proposition made by Context agents, the Exploitation mechanism can generate a feedback that is sent to the set of Context agents. Context agents are the core of the learning. They are responsible of making action proposition based on what they have observed of previous tutor actions. More details about this architecture can be found on previous work (Boes, 2015).
On the rest of this section, we present the behavior of Context agents.

5.3.2 *Context-agents* behavior

The term context in this paper refers to all information external to the activity of an entity that affects its activity. This set of information describes the environment as the entity sees it (Guivarch, 2014). ALEX interacts with a tutor (virtual or human) which performs a set of demonstration. A demonstration consists in the performance of an action under a particular context. Each time an action is performed, ALEX correlates the effect of the performance of this action on the current situation to effects of this action on the environment. ALEX receives a set of signals O from the environment that describes the current situation. Each signal $o_n \in O$ is a continuous value associated to a unique identifier. The identifier is used to discriminate signals and has no semantic value.

ALEX is composed of a set of *Context Agents*. A *Context agent* is a tripartite structure composed of a context description, an action, and an expectation of the utility of the action under this particular context:

$$< context, action, utility >$$

At start, the set of *Context agents* is empty as ALEX possesses no a priori knowledge. *Context agents* are autonomously and dynamically created. *Context agents* receive signals from the environment (from sensors) which they use to characterize the current context. To build its context description, a Context agent associates to each signal *O* from the observation space a set of two bounds $< o_{min}, o_{max} >$. Every time the observation space *O* is included to its context description, a Context agent makes an action proposal. This proposal could be interpreted as "if you do this particular action under this particular context, you can expect this particular utility". At its creation, a context agent is associated to a unique action. The role of a context agent is then to both learn the context description and the utility associated to its action thanks to feedbacks it perceives from its tutor and signals it perceives from the environment. When the tutor is not acting on the system, *Context agents* are responsible of system autonomy. Then they must cooperate to perform an effective control on the device that satisfies the tutor.

Context agents dynamically manage their context description by either reducing or expanding their bounds (figure 5) in interaction with the tutor. Each time the tutor performs an action, *Context agents* observe the tutor activity and compare it to their own action.

Four adaptation processes can occur:

- **Expansion**: When the tutor performs an action that is close to a Context agent's context, and this Context agent proposes the same action, the Context agent can manage its bound to integrate the current situation.
- **Reduction**: When the tutor performs an action that is different to the action proposed by a Context agent, but this Context agent context description include this situation, the Context agent updates its bound to exclude the current situation.
- **Suppression**: By adjusting its bounds, a Context agent can find himself in a situation where $o_{max} < o_{min}$. Such situation produces the destruction of the Context agent.
- **Creation**: When no Context agent proposes the user action (and no Context agent can expand to represent the situation), the system autonomously create a new Context agent to represent the tutor's action.

Bounds are managed by *Adaptive Value Trackers*, a software component that is able to find the value of a dynamic variable in a given space through successive feedbacks. More information on Adaptive Value Trackers can be found on previous work (Guivarch, 2015). The main advantage of this context description is that no semantic on signals is used: it only observes variation of signal values. In parallel to the adaptation of bounds, *Context agents* maintain a utility value. This utility value helps *Context agents* to disambiguate situations where many *Context agents* with different actions propose to perform an action. Utility is then used to determine which the best action to perform is. Utility value is based on Context agent history. The more a Context agent can makes action proposal that is different to the user activity, the more its utility value is low. On contrary, the more the Context agent proposition has been confirmed by user activity, the more the agent is confident in its utility. This value is managed by the function:

$$C_{t+1} = C_t * (1 - 0.8) + F_t * 0.8$$

where C_t is the utility value at time t, and F is the similarity with the tutor where 1 means that the context is proposing the same action than the tutor and 0 means that two actions are different.

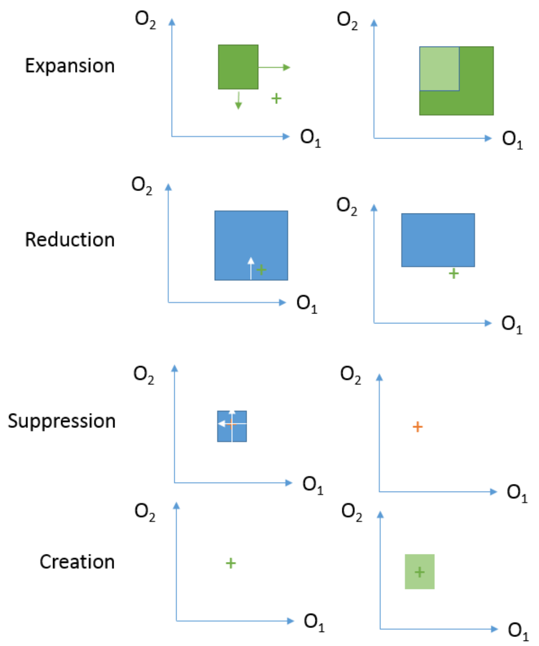


Figure 5: Context agent bounds management example. The cross represents the current situation and its color the action performed by the user.

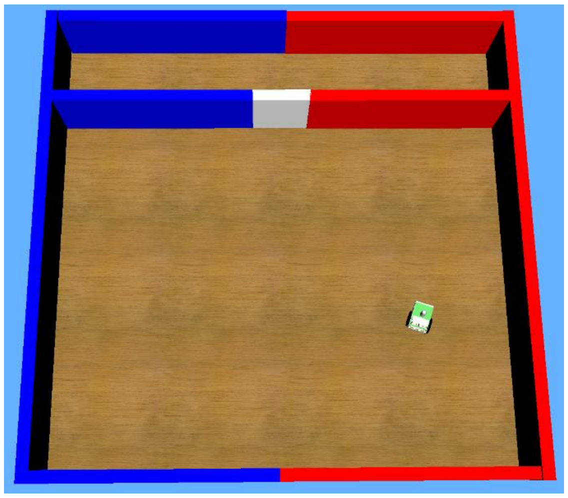


Figure 6: A view of the simulation. The rover evolves in an arena and has to reach the white door.

Whenever the tutor performs an action on the device, *Context agents* use this action to self-adapt by adjusting their bounds and their utility value. However, when the tutor is not acting, *Context agents* use the acquired knowledge to cooperate and control the device.

5.4 ALEX previous work

ALEX has been previously evaluated on a collection task (Verstaevel, 2015). The experiment illustrates the advantages of our approach for end-user, allowing a natural way to express their needs. A tutor controlling a two-wheeled robot demonstrates a collecting task. By comparing the number of artefact collected by the tutor in 5 minutes to the score performed by the rover in autonomy, the results show that ALEX managed to learn to perform the activity more efficiently than the tutor does. The *Context learning* architecture has been abstracted and applied to various domains (Boes, 2014). On this article, we change of viewpoint and want to illustrate advantages of our approach for designers. The next section proposes a use-case study to show those advantages.

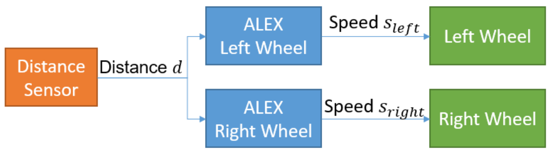


Figure 7: Architecture of the first experiment. Each ALEX receives the distance value and has to associate to this value the adequate speed.

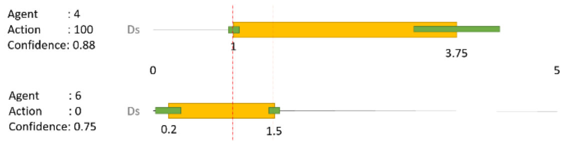


Figure 8: A comparison of two Context agents 4 and 6 extracted from the first experiment. The two Context agents propose a different action under the same context leading to ambiguity.

6 A USE-CASE STUDY

We propose to study the benefits of our approach for the design of the following application:

> A two wheeled rover has to go from an area A to an area B. The passage between the two areas is only possible through a door. The area A may be populated with obstacles. The rover then has to navigate through the area to reach the gate, and then go through it. The experiment is complete when the rover is in the area B and the door is closed.

The experiment is implemented on Webots®, a robotic simulator. The arena is composed of two areas, separated with a white door. The walls on the left part of the arena are blue, the walls on the right part of the arena are red (see figure 6). The ALEX implementation we used is developed in Java and is the same for all experiments. At each time step, an ALEX instance receives data values, the action performed by the tutor and in response, provides the action to be performed.

Each experiment is decomposed in two phases. First, the designer performs a complete demonstration of the activity during which he takes control of the rover. This first phase allows each ALEX instance to acquire Context agents. Secondly, the rover performs the task autonomously by using the previously learnt Context agents. The designer then observes the activity to determine if or not the rover behavior is satisfying.

In case of a failure in the reproduction of the task, the designer can extract and analyze Context agents' structure.

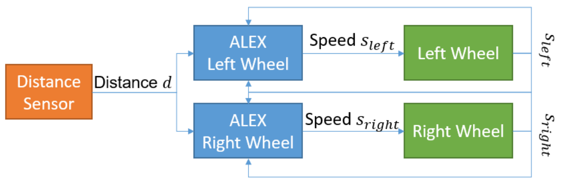


Figure 9: Modified architecture of the first experiment. Each ALEX now receives the distance and the current speed of both wheel.

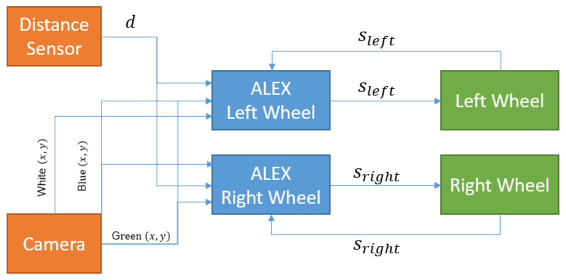


Figure 10: The second experiment architecture. A camera is added providing three new couple of value (x,y) for each detected color.

6.1 First experiment: avoiding obstacles

In accordance with the XS Robotic vision, the designer first identifies the functionalities involved in the application.

> *The rover is composed of two wheels, each wheel controlling its own speed value from -100 to 100. Each wheel is then an XS function of action.*

Then the designer needs to provide to the rover some perception about its surrounding environment.

> *As the task involves avoiding obstacles, a distance sensor is identified as required to navigate through the area. The distance sensor is then an XS function of perception.*

Once both functions of action and perception have been identified, the designer can realize its first experiment and try to teach to the rover the task. The figure 7 illustrates the architecture of this first experiment.

After this first demonstration, we observe that the rover fails in its navigation task. By observing the Context agents inside each ALEX instances, we found that using only distance value leads to ambiguities in the demonstration. The phenomenon is observable in figure 8. The figure shows the structure of two Context agents after the demonstration. The yellow area corresponds to the values of the distance sensor where the Context agent is valid. The green area to the values where the Context agent is extensible. Those two Context agents are extracted from the right wheel ALEX. They propose a different action. The first one propose to go at a speed of 100 whereas the second one proposes to go at a speed of 0. If we observe the two validity range of the Context agents, we observe an overlap. This overlap means that the two Context agents will propose their action in similar situation, leading to ambiguity. The reason is that using only an ultrasound sensor is not well enough to discriminate each situation. When the rover is at a particular distance, it can express either that the rover is approaching an obstacle or moving away.

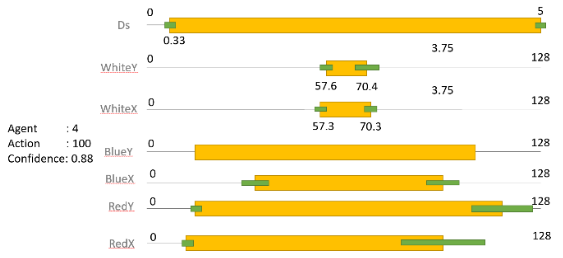


Figure 11: A particular Context involved in the last experiment. This Context agent is a lot more sensitive to the White (x,y) value than the other.

> *To disambiguate those situations, the designer propose to use the current speed value of both wheels as an input to the ALEX instance.*

A new demonstration is performed with this new architecture (figure 9) and the rover now succeed to navigate through the area. However, as it cannot differentiate a wall from the door, the rover fails to learn to reach the door.

6.2 Second experiment: reaching the door

> *As the rover needs to differentiate walls and the door, the designer proposes to add a Camera on the rover to recognize characteristics of the objects to be detected. Using a detection algorithm, the camera can provide visual information about the environment of the rover. As walls and the door have different colors, the camera identifies the coordinate $(\boldsymbol{x}, \boldsymbol{y})$ of the center of each of the three color blue, white and green.*

A Camera is added to the simulation to provide new data to the rover. Each ALEX instance now receives, in complement of the previous data, the coordinate $(\boldsymbol{x}, \boldsymbol{y})$ of the center of each color (see figure 10). If no artefact of one color is detected,

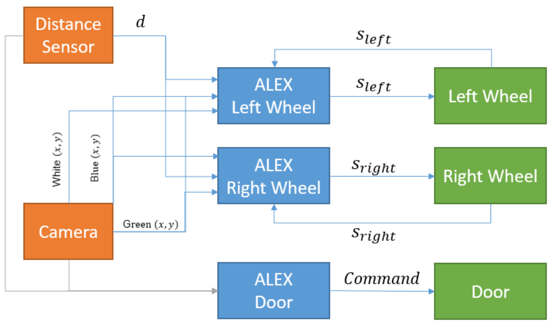


Figure 12: The architecture of the last experiment. The door is now controlled by and ALEX instance and receives the same information than the other ALEX.

the coordinates provided are (-1,-1). The addition of the camera does not involve any modification on the ALEX instances. The tutor then performs another demonstration of the task.

Now the rover manages to navigate inside the arena and reach the door. By observing the structure of the *Context agents* involved in this experiment, we found that the agents involved in the part of the activity reaching the door have learnt to be less sensible to the blue and red coordinates. One example of those agents is visible in figure 11. The validity range associated the signals $\boldsymbol{WhiteX}$ and $\boldsymbol{WhiteY}$ are smaller than the one associated to $\boldsymbol{BlueX}$ and $\boldsymbol{BlueY}$, and $\boldsymbol{GreenX}$ and $\boldsymbol{GreenY}$. As the activity only involves identifying the white door, the other data are unrelated. Our designer can exploit this information to remove unused data from the system. However, the rover failed to complete the task. While it managed to reach the door, the rover failed to open it as its engines were not powerful enough.

6.3 Last experiment: opening the door

For the third experiment, the door is equipped with a motor. An ALEX instance is associated to the door and must learn when it has to be open and when it has to be closed. For thus, the door receives the same data than the two ALEX instances controlling the wheels (Figure 12). Adding this new effector does not involve any action on the pre-existing devices and previously learnt Context agents can be kept.

The tutor performs a final demonstration of the task, demonstrating to each component the desired behavior. At last, the rover managed to learn to reach the area B. The door correlated the action of opening to a low value of the distance sensor signal and the coordinate $WhiteX$ and $WhiteY$ near the center of the screen. The rover and the door managed to collaborate without direct communication. As they share perception, they have enough information to coordinate their activity.

6.4 Synthesis

The whole experiment illustrates both the Extreme Sensitive Robotic paradigm and ALEX capacity to learn in interaction with human. The design of a robotic application with ALEX allows the designer to focus on the desired functionality and to let to ALEX the duty to find correlations between the performing of an action and the state of sensors. An application can be designed incrementally by gradually adding new sensor and effectors. The usage of a learning technic based on self-observation allows the designer to point out situations of ambiguity or situations where some data are useless. Adding a new sensor or a new effector is not a complex task anymore as each component can self-adapt.

7 CONCLUSION

The design of a controller for a robotic application is a complex task. In this article, we propose to give to each device the ability to self-adapt in interaction with its environment. We propose an approach named “*Extreme Sensitive Robotic*” which focuses on the bottom-up design of robotic application. To enable each device to self-adapt, we proposes the use of ALEX, an adaptive multi-agent system based on Context-Learning.

In previous work, we have shown the advantages of our approach for end-user, enabling the robotic device to automatically learn a behavior from demonstrations. In this article, we have taken the viewpoint of the designer of such a system. Through a use case, we have shown that the combination of Extreme Sensitive Robotic and ALEX could help designers to incrementally build their system. By studying the *Context agents* dynamically created by ALEX, the designer can point out ambiguity in signals and decide to increase the perception capacity of the system. The same analysis can lead the designer to add new effectors on the system. As each effector is designed to be self-adaptive, the appearance or disappearance of an effector or of a new sensor does not imply to re-act on the previously deployed effector.

However, the study of *Context agent* is in this paper still hand-performed by the designer. Then, work is being made to allow *Context agents* to automatically discover situations of ambiguity where data are missing and to automatically solve those situations. Such automatic process involves to distribute intelligence inside each sensor which will become a cooperative agent by locally analyzing how they interact with *Context agents*.

REFERENCES

B. D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, *Robotics and autonomous systems* 57 (5) (2009) 469–483

A. Billard, S. Calinon, R. Dillmann, S. Schaal, Robot programming by demonstration, in: Springer handbook of robotics, Springer, 2008, pp. 1371–1394.

J. Boes, J. Nigon, N. Verstaevel, M.P Gleizes & F. Migeon. The Self-Adaptive Context Learning

Pattern: Overview and Proposal. In: Proceedings of the Ninth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT2015), 2015

N. Bonjean, W. Mefteh, M.-P. Gleizes, C. Maurel, F. Migeon, Adelfe 2.0, in: *Handbook on Agent-Oriented Design Processes*, Springer, 2014, pp. 19–63.

R.A Brooks. Elephants don't play chess. Robotics and autonomous systems, 6(1):3-15, 1990.

D. Capera, J.-P. Georgé, M.-P. Gleizes, P. Glize, The amas theory for complex problem solving based on self-organizing cooperative agents, in: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003, IEEE, 2003, pp. 383–388.

Frank Guerin. Learning like a baby: a survey of artificial intelligence approaches. *The Knowledge Engineering Review*, 26(02):209-236, 2011.

M.-P. Gleizes, Self-adaptive complex systems, in: Multi-Agent Systems, Springer, 2012, pp. 114–128.

Guivarch, V., Camps, V., & Péninou, A. Context awareness in ambient systems by an adaptive multi-agent approach. In *Ambient intelligence*, Springer Berlin Heidelberg, 129-144, 2015

S. Lefèvre, A. Carvalho, F. Borrelli, Autonomous car following: A learning-based approach, in: *Intelligent Vehicles Symposium* (IV), 2015 IEEE, IEEE, 2015, pp. 920–926.

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials*, IEEE, 16(1), 414-454, 2014.

Picard G and Glize P, Model and Experiments of Local Decision Based on Cooperative Self-Organization. In: *Second International Indian Conference on Artificial Intelligence* (IICAI'05), 2005.

Rolf Pfeifer and Josh Bongard. How the body shapes the way we think: a new view of intelligence. *MIT press*, 2006.

D. Silver, J. A. Bagnell, A. Stentz, Learning from demonstration for autonomous navigation in complex unstructured terrain, *The International Journal of Robotics Research*, 2010.

Alan M Turing. Computing machinery and intelligence. Mind, pages 433-460, 1950.

N. Verstaevel, C. Régis, M.P Gleizes, F. Robert. Principles and Experimentations of Self-organizing Embedded Agents Allowing Learning from Demonstration in Ambient Robotic. *Procedia Computer Science*, vol. 52, p. 194-201, 2015.

N. Vukovi´c, M. Miti´c, Z. Miljkovi´c. Trajectory learning and reproduction for differential drive mobile robots based on gmm/hmm and dynamic time warping using learning from demonstration framework, *Engineering Applications of Artificial Intelligence 45*, 388–404, 2015.

Weiser, M., The computer for the 21st century. *Scientific american*, 265(3):94–104, 19

J. Zlatev and C. Balkenius. Introduction: Why epigenetic robotics? *Epigenetic Robotics*, 2001.

Development of an algorithm for energy efficient automated train driving

Ozhigin A., Prunev P., Sverdlin V., Vikulina Yu.

Siemens LLC, SEN-RU, 191186, Volynskii lane, 3A, Saint-Petersburg, Russia
e-mail: artem.ozhigin@siemens.com, pavel.prunev@siemens.com,
victor.sverdlin@siemens.com, yulia.vikulina@siemens.com

Key words: *energy efficiency, energy management, automatic train operation, optimization problem, genetic algorithm, smart transportation, real-time systems*

1 Abstract

Automated train driving function is greatly demanded in high-speed and commuter trains operated by Russian railways. Siemens Corporate Technology is involved in the development of such real-time function within a "robotised" train control system. The main intention of the system is not only to relieve the human driver from routine control over traction and brakes (allowing him to pay more attention to assurance of safety) but also to increase train efficiency by reducing the amount of consumed energy. The system under development is intended to be integrated into the existing architecture of Siemens high-speed trains as an additional function. Its environment is constituted mainly of control systems of a high safety integrity level. This on the one hand guarantees that the automated train driving system will not have any impact on train safety, but on the other hand it implies additional restrictions on the operation of the system to be fulfilled. This paper presents the experience in implementation and verification of automated train driving algorithms for Velaro RUS and Desiro RUS trains.

2 Introduction

In recent years the question of energy efficiency has become the gravest it has ever been. Railway operation considerably contributes to the overall energy consumption and, therefore, deliberate railway management systems can enhance the global natural resources savings. Significant advances have been made to improve the efficiency of the train motion: minimization of resistant forces, construction improvements, recuperation capabilities [1]. Nevertheless, there still is opportunity for further savings of energy. In particular, taking into account the track profile, optimization of the driving strategies, measuring of the consumed energy and further analysis to some degree can reduce the cost of trips.

Innumerable studies have contributed to the energy efficiency in train control. A detailed review of works on optimal train control can be found in [2]. Diverse methods and approaches can be found in literature concerning optimal train control, optimal schedule construction and railway system management. Beginning with the pioneer works of Milroy [3], Howlett [4], Asnis et al. [5], numerous scientists developed and investigated train control optimization problem, giving birth to great number of specific methods and approaches. Two different models can be highlighted among them: a model with continuous control [3], [4], [5]; and a model with discrete control [6]. The first model considers applied acceleration as a control variable whereas the second one considers switches of traction and brake for this purpose. The second model has received much attention in recent decades, due to the opportunity to model the real train control directly and to obtain as a solution a sequence of a driver's actions for optimal train control.

One of the most effective approaches to energy efficient driving is speed regulation through coasting control, minimization of brakes or a combination of optimal strategies [7]. In [8] driving strategy 'power-speedhold-coast-brake' is considered and the algorithm of optimal switching points finding is proposed. The "speed regulation without braking" strategy is implemented for efficient manual driving on long distance lines in [9].

Among the methods of solution of the train control problem one can point out two main groups widely used by the researchers: exact methods and meta-heuristical.

The first approach was utilized, for instance, by Howlett in [10]. He has elaborated the algorithm to minimize the energy consumption using Pontryagin's principle and Kuhn–Tucker equations. Assuming that the trip shall be completed with a fixed time he finds local optimal points of control switching for each part of the track with a particular slope and obtains a globally optimal strategy.

In the paper [11] Matsuura investigated an algorithm optimizing a train speed profile by the Bellman's Dynamic Programming (DP), taking into account track profile incline, regenerative brake and so on. The parallel computing technique is introduced to deal with the computation complexity issue. Comparison of Dynamic Programming (DP), gradient method and Sequential Quadratic Programming (SQP) is introduced in [12] in application to the complete optimization of multiple trains' speed profiles and energy storage devices with constraints. However, the increase of control input dimension causes explosion of computation time and use of memory space and the introduced methods cannot be applied to real-time control of trains.

As long as the train motion is a multicriterial process, constructing an analytic optimization algorithm, taking into account all the constraints, is still very difficult. Using Pontryagin's principle of maximum or the Bellman equation requires significant simplifications of the problem considered. Furthermore, these methods reduce the problem to the system of differential or even partial differential equations which need to be solved by means of some numerical methods, e.g. Runge — Kutta methods or finite element method respectively. Therefore, the final solution inevitably contains inaccuracies.

The aforesaid is the reason of the ubiquity of the second group of methods involving diverse metaheuristics: genetic algorithm (GA), simulated annealing, tabu search, etc. Although these approaches do not guarantee to find the global optimum point, in most cases it is sufficient for real-time applications to know the quasi-global optimum. During the last decade a lot of papers were published where that kind of algorithm was used for solving the efficient train control problems ([9], [13], [14], [15], [16], [18], [19], [21], [22], [23], etc.).

The genetic algorithm, for example, was utilized in works of Sicre et al. [9], [13], where the problem of constructing an on-line advisory system for efficient manual train driving is solved. The manual driving is modelled by means of fuzzy parameters as long as the exact moment of control switching depends on the reaction speed of the driver.

The GA-based approach is proposed in [14] to search for a control strategy under different operating conditions and the realistic electromechanical simulation model is utilized for energy consumption calculation. GA is also utilized in the works of Besinovic et al. [15] for calibration of the train input parameters using observed data; De Martinis et al. [16] proposed a framework operating on timetable, rolling stock characteristics, signalling system and infrastructure characteristics and an optimization subroutine composed by the genetic algorithm.

In most works the control sequence is represented by a pre-defined array of positions or as a mesh with fixed size. This evidently causes the loss of generality. In this work we propose an approach where the control switching points are computed during the GA routine, allowing to select not only a type of control, but also a particular switch moment minimizing the consumed energy.

Among the works utilizing other meta-heuristical algorithms we should mention the PhD thesis of Kim [17] with an approach to optimize train motion strategies for various track alignments (single or mixed) and maximum operating speed (constant or variable) using simulated annealing. Modified and hybrid meta-heuristic methods applied to optimal train control were considered in works of Wei et al. [18], Bigharaz et al. [19].

Considering the uncertain disturbances arising from the weather, route and locomotive rolling conditions, Li et al. [20] proved the existence of an optimal operation strategy for stochastic train energy-efficient operation.

The main advantages of meta-heuristic methods are relatively simple logic, the capability to expand the problem and an opportunity to take into account any new circumstances and conditions without affecting the optimization algorithm. These methods do not have requirements of differentiability or continuity on the functions involved. The possibility of mutation in the genetic algorithm allows to not be stuck in the local optimum. Moreover, as a result of computations a set of solutions is obtained and if the best one is not suitable for some reason, another candidate can be taken right away. Meanwhile, analytical methods provide only one solution and in order to satisfy new conditions, the necessity of the recalculation of the whole problem arises. In some cases, reconstruction of the method itself may be needed. This can be impossible or difficult to carry out.

Until recently, the question of efficient train control was a responsibility of the driver, who used his own experience, known heuristics and driving patterns as a key to optimality and efficiency. Autopilot (automated train driving) algorithm providing efficient driving will take the responsibility of routine actions and will allow the driver to concentrate on more important trip factors.

Despite the fact that there are a lot of works devoted to energy efficient train control (see [9], [16], etc.), none known to us deals with completely automated control. Apparently, this is a consequence of the local regulations prescribing that the train must be always under human control as a guarantee of safety. An optimization algorithm can be implemented only as an advisory system. On the contrary, in this work we are trying to develop an algorithm which will be able to implement automated real-time train control capable to realize the whole trip without human interference. The genetic algorithm is applied to the problem in order to find an optimal sequence of controls for energy efficient train driving.

Safety is a crucial matter in transportation systems. As mentioned before, the automated train driving function is surrounded by train control systems of a high safety integrity level and it does not control the motion of the train directly. The commands generated by the system under consideration are sent to train control systems and treated by them as recommendations which will only be implemented if safety is not violated. This way of integration isolates the automated train driving system and prevents any impact on train safety. Due to the isolation there are no real safety requirements imposed on the autopilot. On the other hand, in order to be in actual control of motion and perform designated functions the autopilot shall not violate criteria implied by safety systems for its requests to be implemented. Inappropriate operation of autopilot can affect on passengers' comfort (e.g. frequent switching between traction and braking modes is perceived as not comfortable) and even cause engagement of emergency modes of safety systems (e.g. when autopilot operation causes violation of speed limit). This must be taken in serious consideration in the development of an automated driving algorithm.

The objective of our work is to develop an automatic real-time train control system with a view to reducing of energy consumption. For that purpose we need to find an equilibrium between three antithetic objectives: energy efficiency, fulfilment of schedule constraints and speed restrictions and passengers' comfort.

3 Problem Statement

The aim of this section is to state an optimization problem for energy efficient train control. Firstly, let us consider the mathematical model of the train which is one of the most important parts of a system.

3.1 Train Model Description

A train model is designed to define the velocity and position of a train for each moment of a trip basing on the following parameters: traction intensity (%), brake intensity (%), track profile, adhesion coefficient, weather conditions such as wind velocity. Generalized traction and brake curves were determined from the train documentation.

From the Newton's law we obtain the following dynamic equation of train motion:

$$M\frac{d^2S}{dt^2} = (u_{tr} \;\; u_{br})\begin{pmatrix} F_{tr} \\ F_{br} \end{pmatrix} - F_{fr} - F_h, \tag{1}$$

where M is train mass, S is train coordinate, F_{tr} is traction effort, F_{br} is brake effort, F_{fr} is friction force, F_h is projection of the gravity force, u_{tr} and u_{br} are traction and brake control correspondingly. Traction force values are determined by the train specification. Brake force F_{br} comprises two components: electrodynamic and pneumatic brake effort. Train control system is responsible for actual value of the traction effort and for dynamic distribution of braking efforts between pneumatic and electrodynamic brakes. Control system is trying to fulfil the requests from Autopilot on required speed and acceleration values while simultaneously preventing the violation of safety requirements.

The next term in the eq. (1) is friction force F_{fr} modelled using "Davis equation" [24]:

$$F_{fr} = A + Bv + Cv^2,$$

where $v = \dfrac{dS}{dt}$ is velocity and the coefficients A, B and C are determined by aerodynamic characteristics and rolling resistance. Their values were derived from the train specification. Aerodynamic characteristics include influences of the wind velocity and tunnels. It is possible to take into account other weather conditions, e.g. precipitation, as easy as adding one more term in right-hand side of the eq. (1).

The more sophisticated train models, including non-linear ones, can also be considered and the more realistic electromechanical simulation can be embedded. Complexity of the simulation will not require any changes in the optimization algorithm. Thus, in this work we use the standard simplification in order to pay all the attention to the energy efficiency and finding an optimal speed profile.

Solution of the optimization problem $U(S)$ is represented as a sequence of commands u_i and corresponding switching point s_i.

$$U(S) = [s_i \;\; u_i] = \begin{bmatrix} s_1 & u_1 \\ s_2 & u_2 \\ \vdots & \vdots \\ s_n & u_n \end{bmatrix} \tag{2}$$

Control u_i can be equal to $\{0, 1, 2, 3\}$ where 0 designates the brake, 1 – coasting mode, 2 – cruising mode and 3 – traction. Coasting mode is implemented if traction and brake forces are equal to zero and only friction and gravitational forces act on a train. Cruising mode means the keeping by all means of the current speed using traction or brake force depending on the track profile and other factors. Each mode defines the value of control variables u_{tr} and u_{br}: both equal to zero in coasting mode, one of them has some value from an interval $[0, 1]$ in cruising mode, $u_{tr} = 1$ in traction mode, $u_{br} = 1$ in braking mode.

The train model provides information about energy consumption and recuperated energy. Energy consumed from the beginning of a trip up to the current point, is computed according to the relation:

$$E = \int_0^s F_{tr}\, dS,$$

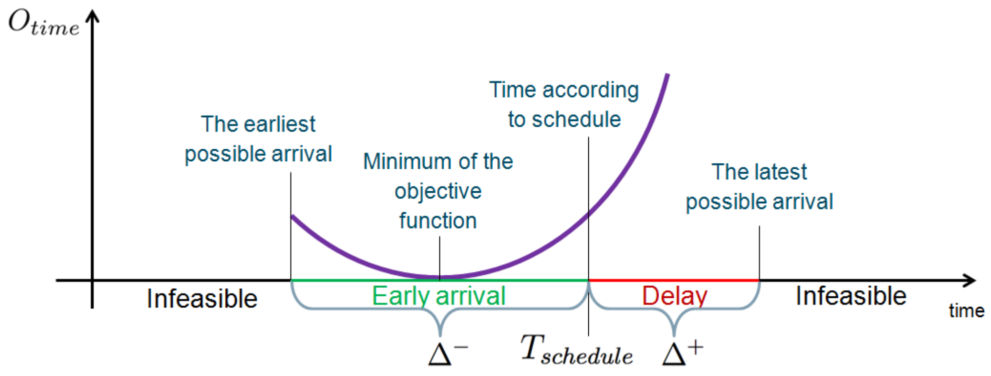


Figure 1: Time component of the objective function O_{time}

where s is current coordinate, E is consumed energy.

Recuperated energy depends only on the electrodynamic brake force which is a component of the whole brake effort as was described above:

$$E_r = \eta \int_0^s F_{edbr}\, dS,$$

where E_r is recuperated energy, η is the maximum percentage of the electrodynamic brake energy which can be returned to the grid, F_{edbr} is electrodynamic brake force.

The regenerative brake opportunity enables to significantly reduce the energy consumption, although, there are some difficulties in efficient utilization of recuperated energy, the first of which is a necessity to absorb the energy by accelerating train or by energy storage device. If the energy is not absorbed, energy consumption is larger because of regenerative failure [12]. However, due to the lack of necessary infrastructure facilities in Russian railways the recuperated energy can not be spent efficiently. Therefore, in this work the regenerative brake energy is not a part of fitness function and is considered as loss.

3.2 Optimization Problem Statement

To complete an optimization problem statement let us define the constraints and restrictions imposed on it. The information defining the train run parameters is described in a digital schedule uploaded before the trip into the automatic train driving system. This schedule contains set of route points with prescribed arrival and departure time, values of speed restriction on different route segments, altitude of the route points. The intention of the system is to make decisions on the driving mode to be implemented at every moment along the route based on current coordinate and schedule data.

Firstly, the optimal train control should satisfy the schedule constraints. We can describe the time condition using the following inequality:

$$T_{schedule} - \Delta^- \geq T \geq T_{schedule} + \Delta^+,$$

where Δ^- and Δ^+ characterise the time tolerance interval, T is trip time for the particular solution, $T_{schedule}$ is time demanded by the schedule.

The boundary conditions define departure and destination points:

$$S(0) = S_{depart}, \quad S(T) = S_{dest},$$

where S_{depart} and S_{dest} are respectively the departure and destination stations coordinates.

Finally, the velocity shall satisfy the speed restrictions:

$$V \leq V_{max}(S),$$

where $V_{max}(S)$ is maximum permitted speed for the coordinate S. Here it is important to note that the length of a train should be taken into account because that constraint should not be violated by any part of a train.

3.3 Objective Function

We understand the term optimal first of all as energy efficient. Nevertheless, the time of the trip should be also included into consideration by an optimization algorithm. Otherwise, the best chosen solution will always

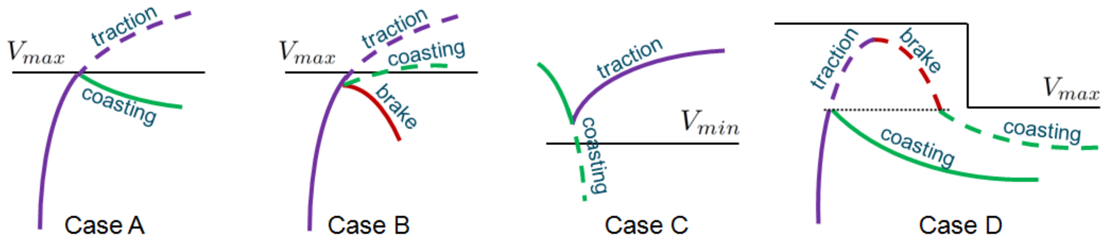


Figure 2: Feasibility check and analytical improvement

implement a trip with longest duration. Therefore, we include in the objective function not only an energy related term, but also a time related one [25]:

$$O = O_{time} + O_{energy} \to \min, \quad O_{time} = \frac{(T_{advise} - T)^2}{(\Delta^- + \Delta^+)^2}, \quad O_{energy} = \frac{E}{E_{flat-out}} + \frac{B}{B_{flat-out}}, \tag{3}$$

where O is objective function, O_{time} and O_{energy} are terms, estimating correspondingly the time and energy optimality.

It is evident that $T = T_{advise}$ provides the minimum for O_{time}, see fig. 1. By default $T_{advise} = T_{schedule} - \frac{\Delta^-}{2}$, but it is to be adjusted by the algorithm of time buffers optimization. This special form of O_{time} gravitates the solution to T_{advise}, not to $T_{schedule} + \Delta^+$ as it would be if only the energy related term in the objective function would be considered.

The energy optimality component consists of two terms. The first is consumed energy E divided by the $E_{flat-out}$ – energy spent during the flat-out trip, i.e. intense mode with minimal trip time. The second one is estimation of brake energy B divided by the one spent during the flat-out mode $B_{flat-out}$. Adding of that component to the objective function allows us to minimize the number and duration of brake modes.

Energy spent in flat-out mode is considered as maximum possible. Although, technically, one can exceed it by using an enormously inefficient solution. Nevertheless, flat-out mode gives us a perfect tool for simple normalization of the energy component of the objective function.

4 Optimization Algorithm

The typical schedule for the trip contains several fixed time points. Some points demand the stop, others – only passing the station within particular time interval defined by tolerances. In order to control the passage time for each point, the optimization process should be separately implemented between all of them instead of considering only points with stops.

For the case when the schedule is unbalanced (too large difference between the route segments' time buffers) we need some preliminary process of estimation and rearranging of the time buffers. For that purpose we move the parameter T_{advise} forward and backward within the tolerance interval for each trip part between two fixed time points.

At the first stage of the optimization process several typical solutions are analysed:

- flat-out solution,
- flat-out solution with cruising modes replaced by coasting modes,
- solution with maximum possible coasting mode duration (maximum time consumption estimation).

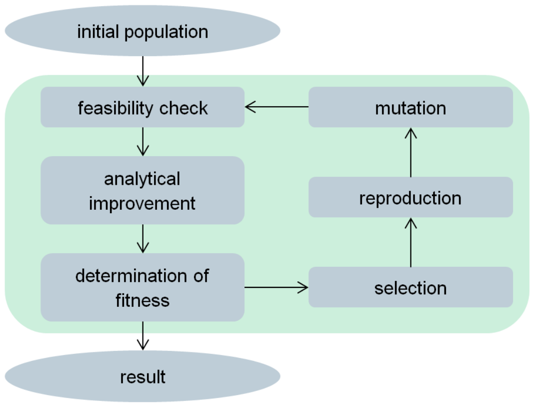


Figure 3: Genetic algorithm chart

If the time buffer after time tolerances optimization is negative, an optimization will not be started and the flat-out mode will be taken as the solution for current trip interval. Otherwise, the time consumption for the next typical solution is checked. If the time buffer is negative, the second typical solution will be taken as a result. If there is a time for improvements, the last check is implemented and if the time buffer for the third typical solution is still positive, then additional brakes will be added to the control sequence during the optimization.

Then, the optimization process based on the genetic algorithm (GA) is started, see its main stages fig. 3. The first step is filling of the **initial population** which consists of the randomly generated solutions. During that process not only controls u_i are generated randomly, but also the moment of switching the command, under assumption that the minimal time between switching the commands equals to 4 sec. This allows to analyse the whole search space of the solutions. Most of the randomly generated solutions are infeasible. But simply discarding the unsuitable solution leads to enormous computation time due to the low value of the relation "feasible solutions / all generated". Therefore a special procedure is necessary to generate feasible random solutions more frequently.

For that purpose, a feasibility check process corrects the solutions if the velocity constraints V_{max} (fig. 2, cases A, B) or V_{min} (fig. 2, case C) are violated. For the V_{max} violation the coasting or brake is added to the control sequence. In case if V_{min} is reached the traction mode is switched on. Secondly, the feasibility check process tracks the time consumed by a randomly generated solution. If the resulting time misses the tolerance interval $[T_{schedule} - \Delta^{-}; T_{schedule} + \Delta^{+}]$, then some random controls are added to the control sequence till the solution becomes feasible or the allowed number of attempts is over. During that process the flat-out solution with minimal time consumption allows to have an estimation of the minimal time needed to get to the destination with current velocity constraints.

Finally, an algorithm of analytical improvement attempts to optimize new solutions using several heuristics, see an example in the fig. 2, case D. Starting from the shortest switches, it iteratively changes the control sequence parts with frequent re-switching traction–brake to coasting mode while the solution is feasible and only if the fitness function has not gotten worse due to that improvement.

As a result, the constructed population is a number of feasible and analytically enhanced solutions represented by the sequence of commands for the train. Then, using a train model described in Section 3.1. we estimate the energy and time consumption for each solution in population. The objective function (or fitness function in terms of the genetic algorithm) is computed according to eq. (3).

The next step is **sorting** of the solutions by fitness and **selecting** of the best or elite members in the population to save them till the next iteration while the worst of them are to be deleted. After that the freed place in the population is filled by the solutions obtained by one-point **crossover**, see fig.4. Here a random mutation can happen, which is in fact an addition of a random control: cruise, traction or coasting or replacing one of the previously defined controls, see fig.6. If the time buffer was too big, together with these commands, an additional brake can also be added during the mutation.

New "children" are checked and corrected by the feasibility check algorithm. From this point the algorithm repeats the iteration until the necessary number of steps is reached.

Further, we continue the optimization for the next trip part and repeat these steps.

At the end of a process we have a set of the solutions which provide the minimum for the objective function. In other words, we have energy efficient train control sequence.

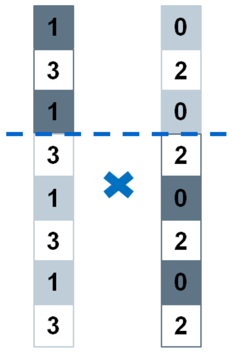


Figure 4: Crossover of two solutions

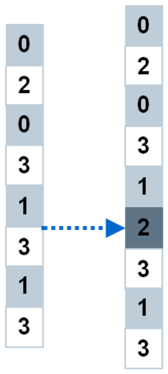


Figure 5: Mutation by adding a control

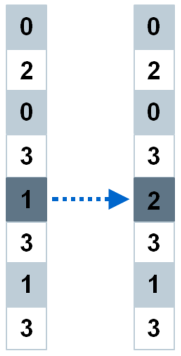


Figure 6: Mutation by replacing a control

5 Results

In order to implement the optimization algorithm described above a C++ program was developed. Estimated time of computations for a 320 km trip is about 40 seconds with Intel Core i5 @ 2.77 GHz, 8 Gb RAM.

An example of the solution for the first 15 minutes of a trip with a real schedule is depicted in the Fig. 7, 8. Despite the fact that the schedule

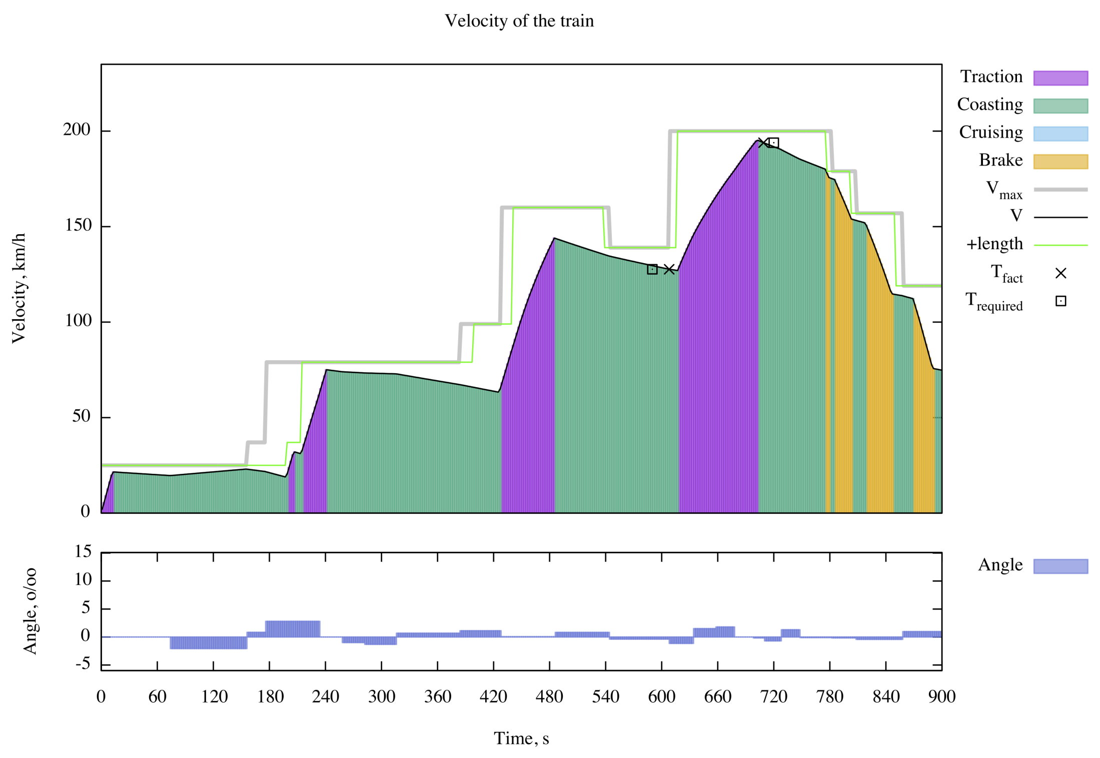


Figure 7: Velocity profile and control modes

is tough, this solution uses the sequences "traction mode → long coasting mode" one after another. Brakes are used only for satisfying the velocity constraints, which is very reasonable. The train is late at the first station (the cross is to the right from the box), nevertheless, it arrives early at the next one (the cross is to the left from the box) without use of the flat-out mode. Cruise mode is not used here at all. Track profile advantages are utilized to hold the speed during the coasting mode: see the interval between 60 and 210 seconds. Here the algorithm does not choose to use brakes or traction, despite the significant change of the speed. The time interval between 420 and 480 seconds also worth mentioning. An algorithm avoids use of a short traction modes here, replacing that with one traction turned on at the end of the 100 km/h constraint and turned off at the middle of the next constraint – 160 km/h. During this speed up the train perfectly fits all the requirements, taking into account its own length (see the green line).

In the optimization process 20 iterations were used. The greater number of iterations would be inefficient because the subsequent steps will lead to diversity reducing in the population whereas no significant improvement of the objective function will be earned. The population size was chosen as a compromise between the computation time and accuracy of the results, i.e. it shall provide the same final solution after multiple experiments. The 200 members population satisfies both requirements.

6 Conclusion

The developed algorithm of automated train control produces an intelligent control sequence which provides an energy efficient trip. The track profile inclines are taken into account, consumed and recuperated energy is computed. There is a capability of consideration of weather conditions and any other additional constraints and circumstances.

We still consider only two possible positions of the traction and brake lever – 0 and 100% intensity. This constraint seems to be reasonable according to the hypothesis that the energy efficient driving means that traction is used for short periods of time but with 100% intensity, where it is possible. However, consideration of several intermediate levels is one of the directions of improvement.

The design of the autopilot with a feature of energy optimality is under discussion and consideration for now. Rough theoretical estimations have shown savings of consumed energy up to 14.5% as opposed to manual driving. Actual values will be computed after the first field trials which are planned for the near future. The authors are looking forward to any reviews and comments.

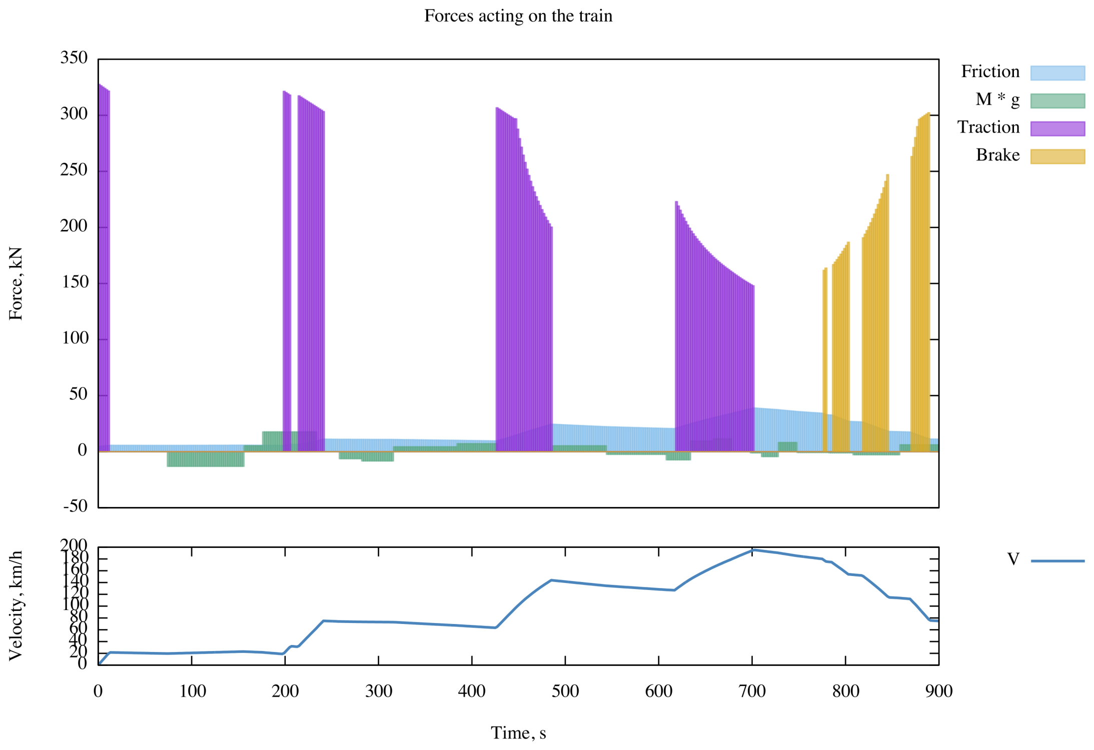


Figure 8: Forces acting on a train

References

[1] Kondo K., 2010. Recent energy saving technologies on railway traction systems. IEEJ Transactions on Electrical and Electronic Engineering 5, 298–303.

[2] Yang J., Jia L., Wei X., 2014. A review on intelligent control for energy-efficient train operation. Proc. of the 11th World Congress on Intelligent Control and Automation, China, 5160–5169.

[3] Milroy, I. P., 1980. Aspects of automatic train control. PhD thesis, Loughborough University, UK.

[4] Howlett P. G., 1984. The Optimal Control of a Train. University of South Australia, Study Leave Report.

[5] Asnis I.A., Dmitruk A.V., Osmolovskii N.P., 1985. Solution of the problem of the energetically optimal control of the motion of a train by the maximum principle. USSR Coput. Maths Math. Phys. 25(6), 37–44.

[6] Benjamin B.R., Milroy I. P., Pudney P.J., 1989. Energy-efficient operation of long-haul trains. In Proc. 4th Int. Heavy Haul Railway Conf., Institute of Engineers of Australia, 369–372.

[7] Howlett P.G., Milroy I.P., Pudney P.J., 1994. Energy-efficient train control. Control Enginneering Practice 2, 193–200.

[8] Albrecht A.R., Howlett P.G., Pudney P.J., Xuan Vu, 2013. Energy-efficient train control: from local convexity to global optimization and uniqueness. Automatica 49(10), 3072–3078.

[9] Sicre C., Cucala A.P., Fernandez A., Lukaszewicz P., 2012. Modelling and optimising energy efficient manual driving on high speed lines. IEEJ Transactions on Electrical and Electronic Engineering 7, 633–640.

[10] Howlett P. G., 2000. The Optimal Control of a Train. Annals of Operations Research 98 (1), 65–87.

[11] Matsuura G., 2014. Optimal train speed profiles by dynamic programming with parallel computing and the fine-tuning of mesh. Computers in railways XIV 135, 767–778.

[12] Miyatake M., Ko H., 2010. Optimization of Train Speed Profile for Minimum Energy Consumption. IEEJ Transactions on Electrical and Electronic Engineering 5(3), 263–269.

[13] Sicre C., Cucala A.P., Fernandez-Cardador A., 2014. Real time regulation of efficient driving of high speed trains based on a genetic algorithm and a fuzzy model of manual driving. Engineering Applications of Artificial Intelligence 29, 79–92.

[14] Boschetti G., Mariscotti A., 2014. Optimizing the Energy Efficiency of Electric Transportation Systems Operation Using a Genetic Algorithm. International Reiew of Electrical Engineering 9(4), 783–791.

[15] Besinovic N., Quaglietta E., Goverde R. M. P., 2013. A simulation-based optimization approach for the calibration of dynamic train speed profiles. Journal of Rail Transport Planning & Management 3, 126–136.

[16] De Martinis V., Weidmann U. A., 2014. The evaluation of energy efficient solutions in train operation: a simulation-based approach. 14th Swiss Transport Research Conference.

[17] Kim K., 2010. Optimal train control on various track alignments considering speed and schedule adherence constraints. PhD thesis, New Jersey Institute of Technology.

[18] Wei L., Qunzhan L., Bing T., 2009. Energy Saving Train Control for Urban Railway Train with Multi-population Genetic Algorithm. Information Technology and Applications 2, 58–62.

[19] Bigharaz M. H., Afshar A., Suratgar A. A., Safaei F., 2014. Simultaneous Optimization of Energy Consumption and Train Performances in Electric Railway Systems. 19th World Congress of the International Federation of Automatic Control (IFAC-2014), Cape Town, South Africa.

[20] Li X., Gao Z., Sun W., 2013. Existence of an optimal strategy for stochastic train energy-efficient operation problem. Soft Computing 17(4), 651–657.

[21] Tuyttens D., Fei H., Mezmaz M., Jalwan J., 2013. Simulation-Based Genetic Algorithm towards an Energy-Efficient Railway Traffic Control. Mathematical Problems in Engineering.

[22] Cucala A.P. , Fernandez A., Sicre C., Dominguez M., 2012. Fuzzy optimal schedule of high speed train operation to minimize energy consumption with uncertain delays and driver's behavioural response. Engineering Applications of Artificial Intelligence 25, 1548–1557.

[23] Carvajal-Carreno W., Cucala A.P., Fernandez-Cardador A., Soder L., 2015. Efficient driving algorithms for non-distributed and distributed trains with the CBTC signalling system. Models and Technologies for Intelligent Transportation Systems (MT-ITS), 418–424.

[24] Rochard B.P., Schmid F., 2000. A review of methods to measure and calculate train resistances. Proc. of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 214, 185–199.

[25] Bocharnikov Y.V., Tobias A.M., Roberts C., Hillmansen S., Goodman C.J., 2007. Optimal driving strategy for traction energy saving on DC suburban railways. IET Electric Power Applications 1, 675–682.

Session 5

Certification

Wednesday 27th, 14:45 – Auditorium St Exupery

Structural Coverage Criteria for Executable Assertions

Cyrille Comar, Jérôme Guitton, Olivier Hainque, Thomas Quinot
AdaCore, 46 rue d'Amsterdam, F-75009 PARIS (France)
{comar, guitton, hainque, quinot}@adacore.com

Abstract

Keywords: Structural coverage, Functional coverage, Coverage criteria, DO-178, Certification, Formal verification, Assertions, Contract Based Programming.

In this paper, we describe some of the issues raised by the use of assertions in software programs that need to undergo coverage analysis for certification purposes, typically in the avionics or railway domains. "Assertions", in this context, designates Boolean expressions expected to always yield True, verifying internal properties of a program at various points of its execution. We can observe an increased use of such constructs in safety critical domains, from more frequent uses of contract based programming techniques.

We explain why we believe that the traditional structural coverage criteria defined for Boolean expressions in certification standards aren't quite adequate for assertions. We propose tracks for possible alternative criteria and examine some of their properties. We also discuss possible viewpoints on the role of assertions within a program and consider some options through which structural coverage on assertions can contribute to the assessment of functional coverage.

This is still exploratory at this stage. Our goal here is to raise attention on the issues and propose initial lines of possible resolutions, as well as various tracks of further work to refine and strengthen the approach.

1 Introduction

A few programming languages promote the notion of *contract based programming* and offer specialized constructs to support this paradigm. A seminal representative of such languages is Eiffel with its *Design by Contract* principle [10], [15]. Ada, and more specifically the Ada 2012 revision of the language, is another example, featuring pre and postcondition aspects, type invariants, as well as new syntactic constructs allowing conditional control within expressions [13].

Programming with contracts is essentially based on *assertions*, Boolean expressions expected to always yield True, embedded in dedicated constructs at various spots of the program. Subprogram pre and postconditions, for instance, embed assertions on properties expected to hold when entering or exiting the subprogram, respectively. Type invariants are assertions that should hold for each object of the type at any time, except potentially while it is being updated.

Such assertions can be viewed in different ways:

They can first be considered as a debugging aid, since they help reduce the distance between the place in the code where an error initially occurs, and the point where it causes an externally noticeable effect and can be detected.

They can also be viewed as a way of clarifying responsibilities: a subprogram precondition expresses the constraints on the kind of situations the routine is able to deal with: it is the responsibility of the caller to ensure that the constraints are respected. Conversely, a postcondition expresses the properties that the routine is guaranteeing, and on which the caller can rely after the call.

Finally, an emerging way of viewing assertions, and in particular pre- and postconditions, is to consider them as a formalization of the requirements of the associated subprogram. For instance, we can express informally the requirements for a square root routine to be: the routine behavior is only defined for positive or null floating point inputs, the square root is the inverse of the square ($square(sqrt(x)) = x$), and we expect a maximum error of 10^{-2} on the result. This can be expressed formally with pre- and postcondition assertions on the subprogram, as in the Ada 2012 example declaration below:

```
function SQRT (X : Float) return Float with
  Pre  => X >= 0,
  Post => abs(SQRT'Result*SQRT'Result - X) <= 0.01;
```

As can be expected, standards for developing critical software such as DO-178C for civil avionics, or EN-50128 for railway, require a fair amount of verification and testing during development cycles. They also require activities showing the coverage and completeness of those verifications. DO-178C defines two families of such coverage objectives:

- *Functional coverage*, which consists in checking that the system behaves as it should, or in other words that all the functional requirements have been verified, and that each of them has been sufficiently exercise to take all aspects of the requirement into consideration;
- *Structural coverage*, which consists in verifying that the program code has been exercised with sufficient exhaustiveness during the testing campaign. Specific structural criteria such as DECISION COVERAGE

or MCDC provide an exhaustiveness metrci based on a close observation of the Boolean expressions that have an influence on program control flow.

This paper explores the implications of conducting coverage analysis on code that uses assertions, starting from the following questions:

- Are the standard structural coverage criteria adapted to code containing assertions?
- Can the structural coverage of assertions help assess functional coverage?

Before discussing these questions, we first need to understand why it can be useful or necessary to keep assertions in the final executable. As a matter of fact, tools usually let the user choose whether assertions will be part of the executable code or not, and it is common practice to enable assertions and other checks of various kinds for testing purposes only, and then to disable them for the final executable to be put in production, in which case pure structural coverage analysis of assertions is less of an issue.

Enabling assertions during testing is a good idea because it makes the testing campaign more efficient: potentially elusive problems such as uninitialized variables or improper aliasing have better chance to get detected. On the other hand, keeping assertions enabled in the final executable is not always the best choice, especially when no recovery mechanism is in place: innocuous errors might cause the program to stop, whereas they would have remained unnoticed in the absence of assertions.

Nevertheless, assertions can be used to provide the guarantee that the program is doing what it is supposed to do, and in some situations it is preferable to stop execution rather than continuing executing a program turned wild. In other words, keeping assertions live in the final system is a legitimate option, and is even necessary in some circumstances, when error recovery is taken care of and it is essential for the program to do *only* what it was designed for and nothing else.

Coverage criteria that make sense for assertions are therefore needed in situations where assertions remain in the executable code, or if they are used to help functional coverage assessment.

DECISION COVERAGE is a common structural coverage metric, and is used for instance when certifiying against level B of DO-178C. It is achieved when each Boolean expression (decision) in a program was exercised to yield both a True and a False outcome. For instance, in an `if` construct, both the `then` and the `else` parts need to be exercised, even if either is empty.

It is tempting to assimilate assertions to `if` statements, and thus consider the asserted expression as a decision. In Ada, for instance, this would mean considering that "`pragma Assert (X);`" is the same as "`if not X then raise Assert_Error; end if;`".

However, this approach is not appropriate in a structural coverage context, because assertions are, by design, expected to never evaluate False, and constructing tests that violate an assertion might prove very challenging, if not entirely meaningless.

When a postcondition represents the requirements of a subprogram, for example, invalidating one consists in trying to find a test that shows that the subprogram doesn't do what it is supposed to do, which should not be possible, and is clearly counterproductive in any case.

The case of preconditions is different: preconditions can be seen as a defensive coding technique, and tests designed to invalidate a precondition could plausibly be part of robustness testing. Such robustness testing could make sense for the preconditions of boundary subprograms that get unqualified inputs from the outside world, or from less trusted code.

However, it is always better to qualify all such inputs explicitly rather than relying on preconditions. Besides, reaching MCDC on preconditions would typically mean finding lots of specific combinations of bad inputs, calling for the construction of lots of artificial cases.

Similar reasoning applies to type invariants, with a stronger contradiction as a result: if we start assuming that an "invariant" may legitimately break during the program execution, there should be requirements stating what to do in this case, and the properties wouldn't be a proper invariant anymore.

As a result, in the majority of cases, DECISION COVERAGE, and MCDC even more so, are not adequate criteria for assertions. This is actually unsurprising since, in essence, assertions aren't decisions by our definition, as they are expected never to evaluate False, whereas decisions are assumed to be legitimately allowed to take both values True and False.

Ignoring assertions for coverage purposes is not a viable option either: even a single assertion might check a wide spectrum of possible cases, and having a means of measuring how extensively this spectrum was exercised by a testing campaign is a necessity. In addition, we believe that some structural coverage on assertions can help functional coverage analysis, when assertions are used to capture subprogram requirements.

We therefore observe that specific coverage criteria to be applied for the coverage analysis of assertions need to be defined. The remainder of this paper identifies possible tracks and illustrates how functional coverage can be helped in the process. The discussion is organized as follows:

In section 2, we propose definitions for three increasingly exhaustive criteria for the coverage of assertions, which could correspond to the three certification level of DO-178C that require coverage analysis. In section 3, we present examples of assertions, and introduce tri-state truth tables that we will use in the following sections for illustration purposes. Sections 4, 5 and 6 provide insights on the way to interpret the definitions on an example assertion, and comment on the sets of tests that can be used to achieve the respective criteria. Section 7 then shows how the definitions allow the nesting of decisions within assertions.

We then examine a few notesworthy aspects of the proposed criteria: section 8 provides a quantitative complexity analysis, giving a notion of the cost associated with achieving each criterion in terms of amount of testing it requires. Section 9 discusses situations where it may be impossible to satisfy some of the criteria, in particular for assertions which contain coupled operands. Finally, section 10 relates our tracks of thought to existing work, before moving to a conclusion where we summarize the most important points and discuss possible perspectives of further investigations.

2 Proposal of specific coverage criteria for assertions

We propose three levels of coverage criteria for assertions. These three levels are intended as companions to the three criteria defined for regular application code in the DO-178C standard (namely Statement, Decision, and MCDC coverage, depending on the certification level).

DO-178C defines the notion of *condition* to designate operands within decisions. We will reuse this term to designate operands within assertions as well.

For reasons exposed later on, when we need to distinguish conditions within a Boolean expression, the only operators we consider are those with short-circuit semantics such as the `&&` and `||` in C, or `and then` and `or else` in Ada, as well as Boolean-valued `IF`-expressions (equivalent to the C ternary operator `?:`). Logical negations have no influence on expression decompositions for our purposes.

A Boolean expression built from elementary Boolean conditions combined with these operators can be modeled as a Binary Decision Diagram (BDD), and an evaluation of such an expression can then be understood as a traversal of this BDD.

Note that Boolean subexpressions may appear within the considered elementary conditions, for example as formal parameters in subprogram calls, or as the expression that conditions a non-Boolean-valued `IF`-expressions. Such nested Boolean subexpressions do not directly determine the traversal of the BDD, and are therefore out the the scope of the coverage analysis for the outer assertion (of course they are subject to a separate coverage discussion in terms of the usual structural coverage criteria, as detailed in section 7).

On this ground, here are the three levels of criteria we propose:

Assertion True Coverage (ATC) The expression as a whole has been evaluated True at least once.

Assertion True Condition Coverage (ATCC) All the expression conditions have been evaluated at least once as part of a complete expression evaluation to True. Different conditions may have been evaluated as part of different outer expression evaluation instances.

Assertion True Path Coverage (ATPC) All the paths leading to a True outcome within the expression's BDD were taken.

The following sections provide more detailed insights on what satisfying the criteria mean, then explore a few properties of interest regarding the criteria. We will be relying on a simple use-case example to illustrate most of the points.

3 Example assertions and tri-state truth tables

To help illustrate various points in the following sections, we will use the following example use-case for assertions: the doors of an elevator are controlled by a simple Ada program, and we focus on the subprograms responsible for locking/unlocking the doors.

An elevator door can be in three possible states:

- **Locked**: the door cannot be opened;
- **Closed**: the door is closed but not locked, i.e. it can be opened by a user;
- **Opened**: the door is opened. It cannot be locked; only a closed door can be locked.

Only two transitions between these states are invalid: going directly from Locked to Opened, and the other way around.

To enforce this constraint, the operations to lock the door is effective only if the door is Closed; in the other cases, it does nothing. A natural `Nb_Errors` indicates the number of errors that have been generated in the operation ; a failure to lock a closed door is one of the possible errors, so at least one error is returned if the final state is not Locked when the original state was Closed. In Ada 2012, this could be expressed by a postcondition:

```
procedure Lock_Door
  (E : in out Elevator_Type;
   Nb_Errors : out Natural)
  with Post =>
    (if Door_State (E'Old) = Closed then
       Nb_Errors >=
         (if Door_State (E) = Locked then 0
          else 1)
     else
       (Door_State (E) = Door_State (E'Old))
          and then Nb_Errors = 0);
```

For the sake of the example, the unlock operation is not quite symmetrical: trying to unlock a door that is not locked also results in an error, which allows a simpler postcondition:

```
procedure Unlock_Door
  (E : in out Elevator_Type;
   Nb_Errors : out Natural)
with Post =>
  ((Door_State (E'Old) = Locked
      and then Door_State (E) = Closed)
    or else Nb_Errors > 0);
```

In both cases, postconditions express the invariants that the corresponding procedure bodies enforce. Forcing them to False during the test campaign would mean deliberately introduce bugs in the procedure code, which would be purely artificial. Keeping them in the deployed application would still be useful: the violation of any of these assertions means that the program cannot guarantee the safe use of the elevator anymore and the system would stop it and switch to a safe recovery mode: e.g. stop the elevator where it is and call for manual maintenance.

To help illustrate our criteria on this example, we will resort to tri-state truth tables to synthesize the way expressions get valued. The True and False possible values of the conditions or of the expression outcome will respectively be denoted with “T” and “F”, and condition columns have an extra “x” possible state for situations where the condition evaluation is short-circuited. Each line of the table is called an *evaluation vector* and is assigned a unique number that can be used to designate it later on.

For the `Unlock_Door` postcondition, using abbreviations like *Pre_Locked* to denote “the door state on entry was Locked“, *Closed* to denote “the door state on exit is Closed”, and *Error* to denote “At least one error has been returned”, we have an expression with three conditions of the following form and its associated tri-state truth table:

```
(Pre_Locked and then Closed) or else Error
```

#	*Pre_Locked*	*Closed*	*Error*	Outcome
1	F	x	T	T
2	T	F	T	T
3	T	T	x	T
4	F	x	F	F
5	T	F	F	F

4 ATC user level characterization

From the truth table of our example assertion, any of the vectors #1, #2 or #3 is enough to satisfy ATC on its own, as they all yield a True outcome for the expression as a whole.

This is pretty weak on the functional coverage front. The other two criteria are stronger and deserve each a more elaborate discussion:

5 ATCC user level characterization

ATCC is explicitly referring to *complete* expression evaluations as those contributing to the criterion fulfillment. This denotes evaluations that terminate without unexpected interruptions, typically by possible exception occurrences during the tests. Indeed, evaluations that don't terminate yield no value for the expression as a whole, so we can't tell whether the assertion was satisfied. These are implicitly excluded by ATC already. The important extra point about ATCC that deserves being explicit is that we consider the incomplete set of valued conditions as not contributing to the criterion.

To illustrate what sets of test may be used to achieve ATCC, let's consider the `Unlock_Door` postcondition and the associated truth table again.

We can see that evaluation #2 yields True and evaluates all the conditions, so is sufficient to achieve the criterion on its own. #1 alone is not enough since the evaluation of *Closed* is short-circuited, and #3 alone is not enough either since the evaluation of *Error* is short-circuited. The definition allows the combination of #1 and #3 to fulfill the criterion, still, as they both yield True and evaluate all the conditions overall.

This is immediately stronger than ATC from the functional standpoint:

Indeed, a single test vector allows satisfying the criterion alone: vector #2 which corresponds to *Pre_Locked* True, *Closed* False and *Error* True. This is a very interesting case, which checks that *Error* is correctly set on a failed legitimate attempt to unlock a door.

If testing doesn't check this specific case, it has to check two cases as a counterpart; one verifying that *Error* is correctly set on invalid attempts to unlock a door that was not locked on entry (vector #1), and one verifying a situation where the door state transitions from Locked to Closed as expected in nominal conditions (vector #3).

6 ATPC user level characterization

The use of the expression BDD for ATPC makes it potentially hard to understand from a user point of view. This is where focusing on operands combined by short-circuit operators helps. Indeed, in this case the set of BDD paths leading to a True outcome correspond to the set of lines with a True outcome in the tri-state truth table, so figuring out which tests need to be exercised to fulfill the criteria is straightforward from the table.

For our example postcondition on `Unlock_Door`, we can immediately see from the associated truth table that achieving ATPC requires 3 tests, going through all of the first three vectors, which are the ones for which the expression evaluates True.

The gain in strength of functional coverage compared to ATCC is significant, as the criterion requires exercising all the cases of relevance.

Incidentally, however, we notice that *Error* is not evaluated as part of vector #3. It might seem surprising not to evaluate a possible error condition in a vector that contributes to a structural coverage criteria, and here could be an indication that the postcondition might be imprecise in conveying all the functional requirements and might need to be improved.

This last observation, together with the previous comments, illustrates that thinking in functional terms as part of a structural analysis process can be meaningful and of real interest. It so appears that structural coverage analysis on contracts can be of significant potential value for functional coverage discussions.

7 Decisions within assertions and the influence of coding styles

While assertions aren't decisions by definition (an assertion is assumed to only ever evaluate to True throughout testing, whereas a decision can be expected to evaluate True and False over successive tests), assertions can embed nested decisions that might need to be analyzed on their own. This influences the amount and kind of testing required, so is useful to keep in mind when reasoning about the meaning and characteristics of criteria. Here, we present an example to illustrate the point and show how differences in coding style can influence the testing requirements as well.

Consider the postcondition of the `Lock_Door` subprogram in our example. The assertion is of the form:

```
(if A
  then Nb_Errors >= (if B then 0 else 1)
  else C and then D)
```

The IF-expression at the top level is Boolean-valued, so we can construct a corresponding Binary Decision Diagram for the assertion, and apply ATPC on that diagram (fig. 1).

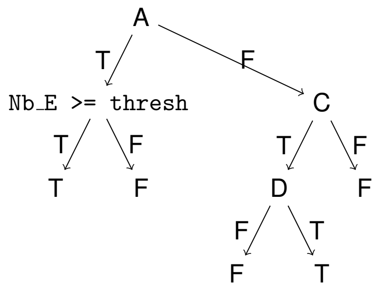


Figure 1: BDD for `Lock_Door` postcondition

On the other hand, the inner IF-expression is not Boolean-valued, and so the corresponding two cases do not directly appear as nodes in that BDD.

Note that short-circuit Boolean operators `and then` and `or else` are really but a specific cases of the more general Boolean-valued `IF`-expression construct: `A and then B` is equivalent to `(if A then B else False)`, and `A or else B` is equivalent to `(if A then True else B)`. This construct exactly captures the elementary building block in BDDs, which model the way decisions actually are evaluated in generated code: the condition at each step selects a path ultimately leading to a True or False outcome. In assertion contexts, paths leading to False are irrelevant, and do not play a role in the coverage analysis.

Conversely, a non-Boolean-valued `IF`-expression just selects a value that will intervene in some other outer expression such as a relational operator or function call: either direction of the condition might yield a True or False outcome; neither direction can be eliminated in an assertion context on the basis that only True outcomes are relevant. Consequently, the relevant coverage criteria for an expression that controls a non-Boolean `IF`-expression are the usual DECISION COVERAGE and MCDC.

8 Complexity assessment

An interesting question is how many evaluations (tests) are needed to satisfy our criteria. The regular MCDC criterion, for example, is known to require no more than N+1 tests for a decision with N conditions. What would be an upper bound for assertions?

For ATCC, the complexity is also linear: no more than N tests are required for an assertion with N conditions. Indeed, for any well-formed assertion, each condition is reachable and from this condition there exists a path to outcome True in the BDD. So for each condition we can provide a test that evaluates the considered assertion and exits on outcome True. Taking one such test per condition gives N tests.

However, for ATPC, the complexity can be exponential, depending on the topology of the BDD. In the study of equivalence between object branch coverage and MCDC that [7] provides, a pathological case was presented where three evaluations were enough to cover an arbitrarily complex decision for object branch coverage; the same case also shows an exponential complexity for ATPC. Consider the following set $\{E_n\}_{n\in\mathbb{N}}$ of Boolean expressions:

- let E_0 be a simple condition expression, with its condition denoted C_0; then define:
- $\forall\ n > 0, E_n = (E_{n-1}$ `and then` $C'_n)$ `or else` C''_n, with C'_n, C''_n independent from each other and from any condition in E_{n-1}.

Figures 2(a) and 2(b) show the BDD of E_1 and E_2. For $n > 1$ it can be seen there that the two True outcomes of E_{n-1} both reach C'_n in E_n and have then two different paths to True; so the number of paths to True in E_n is more than twice the number of paths to True in E_{n-1}. This means that one would need at least 2^n tests to cover E_n for ATPC.

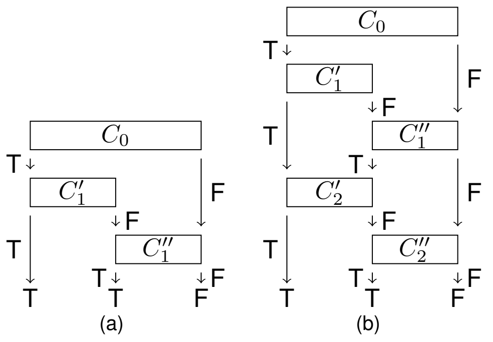


Figure 2: Exponential complexity for ATPC

This example illustrates that exponential complexity comes from multi-path nodes (nodes with more than on predecessor in the BDD). On the contrary, when the BDD is a tree, the complexity is linear: the number of tests needed to cover it for ATPC is at most equal to the number of conditions. This is also the exact amount of tests needed in the case of an assertion that contains only `or else` operators.

[7] provides a case study on two industrial projects showing that multi-path nodes in BDDs are quite rare (less than 1% of decisions). A similar case study would be needed for assertions, but this suggests that assertions where the complexity of ATPC would explode are likely quite rare. In other words, we expect BDDs to be trees in the vast majority of cases, and ATPC is comparable to MCDC in this configuration as covering an expression for ATPC is exactly the same as taking only the True valuations of a MCDC coverage.

9 Achievability considerations

A known issue with coverage criteria on Boolean expressions is the potential inability to achieve coverage in the presence of coupled conditions. This is in particular what lead to the refinement of Unique Cause MCDC into Masking MCDC. The same question holds for coverage criteria on assertions.

Amongst the criteria that we defined in this paper, only ATPC can be impacted by coupled conditions. In the general case, it is possible to build assertions with coupled conditions on which ATPC would not be achievable. The question is whether these cases could be found in a real industrial system.

First, it must be noticed that such cases do not make sense in the absence of multi-path nodes. Indeed, with a tree BDD, not being able to cover a particular path to True means that one of the conditions cannot be changed, which would mean that it is useless and that the assertion is ill-formed.

On the other hand, as seen in the previous section, multi-path nodes in decisions seem to be quite rare in industrial application. To have an impact on achievability, these rare expressions also have to contain coupled conditions, and these coupled conditions have to forbid a path to outcome True. This makes this problematic cases even less likely to appear in real-life applications. It would therefore be manageable to handle these with justified exemptions without increasing too much the workload of the tester.

In that context, we have identified one recurring pattern that requires specific attention. It consists in the elaboration of an assertion that distinguishes different subsets or states, for each of which a separate expression must evaluate True, such as:

Case 1 : expression B must be True;

Case 2 : expression C must be True.

An example of this pattern is the postcondition of operation `Lock_Door` in the elevator example. In this postcondition, Case 1 is when the door is closed before hand and Case 2 is the other cases.

Depending on the way the assertion is expressed, some of the paths may be impossible to exercise in which case ATPC would be unachievable.

The problem lies in an implicit coupling between the predicates denoting each case. Indeed, in general the cases distinguished in such a requirement are mutually exclusive.

If the above requirement is formally expressed in disjunctive normal form (DNF):

```
(Case_1 and then B)
  or else
(Case_2 and then C)
```

then ATPC cannot be achieved, because the implicit coupling makes it impossible to exercise some paths in the expression (which involve `Case_1` and `Case_2` both being True). In this example, the expression's truth table contains:

Case 1	B	Case 2	C	Outcome
T	F	T	T	T

and ATPC cannot be achieved because there's no way to get both `Case_1` and `Case_2` true as part of the same evaluation. In BDD terms, the path outlined in figure 3 can never be taken.

For MCDC, the issue of coupled condition has been handled by refining the coverage criterion into the Masking MCDC variant; in the case of assertions, something like a masking ATPC could be defined formally but this criterion would be harder to understand by testers. Here, we would consider an alternative solution: rewrite the assertion to eliminate multi-path nodes.

Indeed, note that in the context of formalizing the discussion of two mutually exclusive states, the unreachable path does not make any sense, as it first takes a branch denoting the first case, and then falls through to examining the second case. This is a consequence of the DNF formalization failing to capture the notion that `Case_1` and `Case_2` form a partition of the state space.

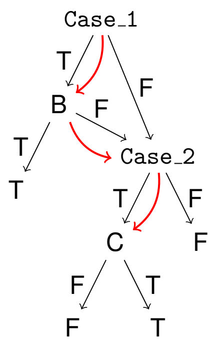


Figure 3: BDD for `(Case_1 and then B) or else (Case_2 and then C)`

Conversely, using a notation that does capture this fact yields a BDD where this nonsense path does not exist. For example, using Ada 2012 conditional expressions, the discussion of two cases can be denoted as:

```
if Case_1 then B elsif Case_2 then C
```

and this time the BDD is a straightforward tree, as depicted in figure 4.

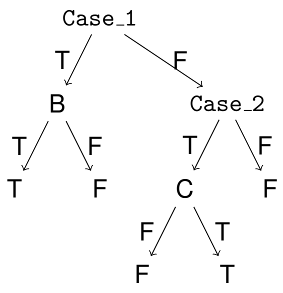


Figure 4: BDD for `if Case_1 then B elsif Case_2 then C`

This notation retains the semantics that the two cases discussed are mutually exclusive: if Case 1 is examined, there is no point in examining Case 2. With this additional semantic information retained, ATCC and ATPC become equivalent in this case, and are reached using two tests:

Case 1	B	Case 2	C	Outcome
T	T	x	x	T
F	x	T	T	T

In addition, when the discussion of cases is thus expressed as a chain of conditional expressions, ATPC can be simply understood as exercising each of the distinct cases, or states. In other words, when a requirement consists in a partition of input space, and the requirement is expressed with a notation such as the above that preserves the notion of that partitioning, then ATPC essentially consists in separately covering each subset in the partition.

Note that this is precisely the form that has been presented for the postcondition of `Lock_Door`. The SPARK 2014 language also introduces a specific notation for this pattern in the form of the `Contract_Cases` aspect, that could have been used as follows:

```
procedure Lock_Door
  (E      : in out Elevator_Type;
   Failed : out Boolean)
with Contract_Cases =>
  ((Door_State (E) = Closed) =>
     Door_State (E) = Locked
     or else Error,
   (Door_State (E) in Opened | Locked) =>
     Door_State (E) = Door_State (E'Old)
     and then not Error);
```

In addition to expressing a postcondition equivalent to the above chained conditional expressions, this also states that the set of cases are both disjoint or complete.

10 Related work

As the coverage level required on Boolean expressions is a typical key differentiator between assurance levels in various certification standards, there are lots of publications treating of Boolean expressions in coverage analysis contexts.

The certification standards themselves, such as [16], [17] for civil avionics, or [4] for railway, are of course a primary reference. A very large number of related papers and documents were written over the years as the experience of using the criteria in real projects built up. Here are just a few examples: [3] clarifies some aspects regarding possible interpretations of the DO-178CB standard, specifically on the notion of “decision”. [5] defines possible variants of the MCDC criterion, amongst which *masking* MCDC was eventually accepted as valid alternative in avionics projects to handle issues with coupled conditions [2]. [12], [11] illustrate what the various criteria can mean in practice from a user or tool qualification perspective. [6] performs a thorough study of MCDC’s main characteristics and relationships with other criteria. [19] proposes a formalization of coverage criteria in Z, and other authors proposed improvements to the traditional decision related criteria, improving their problem detection strength while remaining linear with the complexity of decisions [21], [20].

All these revolve around the notion of *decision* in DO-178C parlance, however. To our knowledge there is no prior publication on the specific set of issues that this paper proposes to explore, with a focus on assertions and their possible connection with functional coverage.

Parallels of interest are nevertheless possible with some connected topics.

In particular, if we consider coverage analysis as a mean to determine if a testing campaign exercised enough of a program logic, a parallel is possible between the concerns we address here and research on the testing complements to formal methods. Indeed, there is an strong duality between aiming at the definition of a useful subset of tests needed to exercise a component correctly, based on formal representations, and verifying that enough testing was achieved thanks to assertion coverage analysis on the implementation code.

[1] and [18], for example, introduce a framework where a formal representation of a component in Z is used to derive partitions of its *Valid Input Space* (VIS) so a single test within a given partition is representative of the entire partition. If one translates the VIS partitioning rules as preconditions, then a set of tests derived from the partitioning process should in principle match or at least encompass the set of tests required to achieve ATPC. As another example [9] presents techniques based on expression DNF to automate the partitioning and test sequencing processes.

11 Conclusion and perspectives

In this paper, we discussed the structural coverage analysis of assertions (Boolean expressions expected to always yield True at various points of a program), and the necessity for specific coverage criteria. We explained why we believe such an analysis is of use in an application certification context, and proposed the definition of three coverage criteria as a basis for further discussion.

One aspect of interest is the idea that some categories of assertions can be understood as a formalization of a program's functional requirements. We have provided examples that show how structural analysis on contracts can be interpreted in a meaningful way in functional terms, and thus provide a valuable basis for a functional coverage analysis.

We believe there is a lot of room for further work on this topic, in particular to refine the definition of relevant coverage criteria. Gathering more data from real use cases on industrial projects would help assessing their actual usefulness, and would certainly foster improvements. It is also quite possible that defining several families of criteria could be relevant to address different categories of assertions, within the context of contract based programming. For instance, we can perceive that preconditions are very different from postconditions, and might thus warrant distinct criteria.

Finally, *assertion coverage* turns out to be a common notion in the hardware design verification area, in association with so-called *Assertion Based Verification* (ABV) systems [8]. It might be interesting to investigate if some of the ideas that led to the definition of an entire methodology in the hardware domain could apply to the software area. This would be kind of a symmetrical idea of the one used in [14], where the authors note that a categorization of software faults works pretty well in the microprocessor validation domain as well, and leverage this similitude.

References

[1] David A. Carrington and Phil Stocks. A tale of two paradigms: Formal methods and software testing. In *Z User Workshop, Cambridge, UK, 29-30 June 1994, Proceedings*, pages 51–68.

[2] CAST, Certification Authorities Software Team. Rationale for accepting Masking MCDC in certification projects. Position Paper 6, August 2001.

[3] CAST, Certification Authorities Software Team. What is a Decision in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)? Position Paper 10, June 2002.

[4] CENELEC. Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. European standard EN 50128:2001, Brussels, Belgium, Mar 2001.

[5] John J. Chilenski. An Investigation of Three Forms of the Modified Condition/Decision Coverage (MCDC) Criterion. Technical Report DOT/FAA/AR-01/18, April 2001.

[6] John J. Chilenski and Steven P. Miller. Applicability of modified condition/decision coverage to software testing. volume 9, issue 5 of *Software Engineering Journal*, pages 193–200, September 2004.

[7] Cyrille Comar, Jerome Guitton, Olivier Hainque, and Thomas Quinot. Formalization and Comparison of MCDC and Object Branch Coverage Criteria. In *ERTS (Embedded Real Time Sofware and Systems Conference)*, May 2012.

[8] Anat Dahan, Daniel Geist, Leonid Gluhovsky, Dmitry Pidan, Gil Shapir, Yaron Wolfsthal, Lyes Benalycherif, Romain Kamdem, and Younes Lahbib. Combining system level modeling with assertion based verification. In *6th International Symposium on Quality of Electronic Design (ISQED 2005), 21-23 March 2005, San Jose, CA, USA*, pages 310–315, 2005.

[9] Jeremy Dick and Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Proceedings of the First International Symposium of Formal Methods Europe on Industrial-Strength Formal Methods*, FME '93, pages 268–284, 1993.

[10] ECMA. *ECMA-367: Eiffel analysis, design and programming language*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, June 2006. 2006.

[11] FAA, Federal Aviation Administration. Software Verification Tools Assessment Study. Technical Report DOT/FAA/AR-06/54, June 2007.

[12] Kelly J. Hayhurst, Dan S. Veerhusen, John J. Chilenski, and Leanna K. Rierson. A Practical Tutorial on Modified Condition/Decision Coverage. Technical Memorandum NASA/TM-2001-210876, NASA, May 2001.

[13] ISO. *Information Technology – Programming Languages – Ada*. ISO, December 2012. ISO/IEC 8652:2012(E).

[14] Sreekumar V. Kodakara, Deepak A. Mathaikutty, Ajit Dingankar, Eep Shukla, and David Lilja. A Probabilistic Analysis For Fault Detectability of Code Coverage Metrics. In *proceedings of the 7th International Workshop on Microprocessor Test and Verification (MTV'06)*, 2006.

[15] Bertrand Meyer. Applying ”design by contract”. *Computer*, 25(10):40–51, October 1992.

[16] RTCA. Software considerations in airborne systems and equipment certification. RTCA document DO-178B, 1992.

[17] RTCA. Software considerations in airborne systems and equipment certification. RTCA document DO-178C, January 2012.

[18] Phil Stocks and David A. Carrington. A framework for specification-based testing. *IEEE Trans. Software Eng.*, 22(11):777–793, 1996.

[19] Sergiy A. Vilkomir and Jonathan P. Bowen. Formalization of software testing criteria using the z notation. In *Computer Software and Applications Conference (COMPSAC)*, pages 351–356. IEEE Computer Society, 2001.

[20] Sergiy A. Vilkomir and Jonathan P. Bowen. From MC/DC to RC/DC: Formalization and Analysis of Control-Flow Testing Criteria. Technical Report SBU-CISM-02-17, South Bank University, CISM, London, UK, 2002.

[21] Sergiy A. Vilkomir and Jonathan P. Bowen. Reinforced Condition/Decision Coverage (RC/DC): A New Criterion for Software Testing. In Didier Bert, Jonathan P. Bowen, Martin Henson, and Ken Robinson, editors, *ZB2002: Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science*, pages 295–313. Springer Verlag, 2002.

MIMOSA: Towards a model driven certification process

Pierre Bieber, Frédéric Boniol, Guy Durrieu,
Olivier Poitou, Thomas Polacsek, Virginie Wiels
ONERA, Département Traitement de l'Information et Modélisation
2, avenue Edouard Belin BP74025, 31055 TOULOUSE Cedex 4
{firstname.lastname}@onera.fr
Ghilaine Martinez, DGA TA, firstname.lastname@intradef.gouv.fr

27-29 JANUARY 2016

Abstract

A certification process usually consists in analyzing, in a restricted amount of time a, potentially very large, set of documents that are intended to convince the auditor that the documented system fulfills all its requirements. The MIMOSA Project presented in this paper introduces a model driven certification process based on the key concepts of argumentation step, patterns and composition. The aim is: at first, to structure the documentation provided as evidences of the good properties of the system, and then to check this structure against identified argumentation patterns that will help identifying lacks or misuse of elements. Argumentation step and composition principles as well as a set of patterns for arguing about real-time properties are given along with their expression in a prototype tool, that offers to describe the architecture, requirements and argumentation in a common language and then offers to compute some basic checks on the argumentation structure.

Keywords : Certification, Safety, Real-Time, Model-based System Engineering, Argumentation

1 Introduction

The MIMOSA[1] project aims at building a frame of reference for the certification of military embedded architectures. The goal of this frame is not to design and develop architectures, but rather to formalize requirements for this kind of architectures and place in front of them acceptable means of compliance, building a coherent argumentation. This frame of reference may be used by DGA to assess architectures proposed by industrial companies with respect to certification standards [5].

The frame of reference includes models of the fundamental concepts of a modular architecture, models of the requirements attached to these architectures and a model of the argumentation of the compliance of the architecture to the requirements. The frame includes different levels of description from functional to hardware; different facets such as Architecture, Safety and Real-time and different concerns Architecture documentation, requirements and argumentation.

Section 2 presents the modeling approach, language and tool. Section 3 describes the argumentation modeling principles, section 4 gives some examples and section 5 concludes.

2 Language Overview

The MIMOSA framework is organised as a three dimensional grid having as dimensions:

- the layer – or level of detail –(function, software, topology, hardware),

[1]MIMOSA stands for Means of engIneering for MOdelling and analysis of modular embedded aeronautic Systems and Architectures

- the concern (architecture, requirement, argumentation) and
- the facet (general, safety, real-time).

Facets are a way to focus on domain specific considerations: properties and associated elements of description. For example, the realtime facet extends the general one by adding:

- specific concepts or attributes to describe an architecture from the realtime aspect such as best and worst-case execution times of each application of the software level,
- specific realtime requirements such as "Worst Case Latency of function f should be bounded by time t" at the functional level and
- specific argumentation pattern for example to efficiently convince that an application worst case response-time is bounded.

Two specific facets are currently handled by the Mimosa framework: safety and real time.

2.1 MIMOSA language internals

The language in which all of those descriptions are expressed should make available to its users concepts such as a function, an application, a binary code and so on. We call this high level language the user language. To be able to fully support the user we need some reasoning capabilities on this user language. Obtaining such capabilities directly on a very rich language as the one we were about to define a very complicated task and leads to hard to maintain result. To obtain such capabilities we then started by defining a simpler language, a formal specification language, called WEIRD, that offers reasoning capabilities by being translatable to propositional logic. Then, the user language is described using WEIRD then inheriting its reasoning capabilities.

Elements of the WEIRD language are *entity*, *concept* and *relation* as well as constraint expressions (with quantifiers). WEIRD offers a typing system that offers to express that an *entity* e is an instance of a *concept* c, or that a *concept* sc is a sub concept of a *concept* c. Another central notion of WEIRD is the notion of *World* that allows modular modeling by constraining visibility and extent to which properties must apply. A World can contain Concepts, Entities, Relations and Constraints/properties to be satisfied. A World can derive from one or several other Worlds; in that case, it has access to all the elements and shall satisfy all the constraints of the World from which it derives.

The notion of World is used to model facets, layers and concerns. In particular one world is associated to each layer (function, software, topology, hardware) as well as to some combination of layers. Those "combination world" mainly contain allocation relations and constraints that take place between elements of different worlds. For example a constraint like "every partition (topological concept) is allocated on a unique CPIOM (hardware concept)" will be expressed in the topological hardware mapping world.

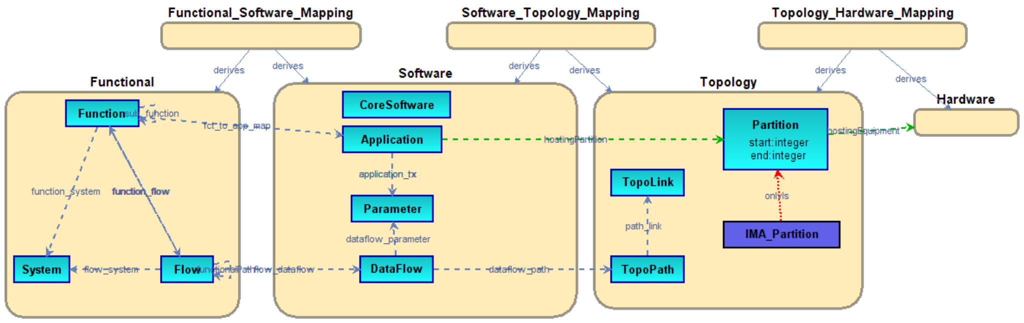


Figure 1: The architecture layers and mapping worlds[2]

This way the final "user language" – the language from which the final user will take elements to describe the system under analyses – is defined. Having this formal WEIRD language offers both the ability to make some reasoning and to ease evolution of the user language.

[2]The content of hardware world have been hidden since it would not have fit in the page width.

In our prototype, WEIRD is translated into propositional logic by applying rewriting rules (translation is rather direct from constraints expressions, user defined relations are kept as relations, all typing information becomes relations, world are used to restrict the domains on which quantifiers are expanded). This way, properties to check are valued by confronting them to knowledge expressed by the user[3].

2.2 Description of the system to analyze

When describing an instance to analyze, newly introduced worlds will derive from the layer, facet and concern worlds for which they are relevant. For example a world describing the functional safety requirements of the examined solution will derive at least from the three corresponding worlds Functional, Requirement, and Safety. This way:

- it will gain access to all the concepts, relations and entities they introduce and
- it will have to respect constraints introduced by all of them.

This derivation may be indirect in some cases: *instance functional description world* will derive from *Functional*, then the above described world may derive from the *instance functional description world* to gain access, at the same time, both to general rules from Functional (indirect derivation) and instance elements and rules from its instantiated version (direct derivation).

These derivations offer to reuse defined concepts and entities in the new worlds as in the example in Figure 2 of a Fire Control function (partial) description. The global CdT function, as well as its enabling applications are directly introduced without the need of redefining anything. In the same way 4 partitions are introduced in the TopoRef world by reusing the concept of IMA_Partition from the generic Topology one.

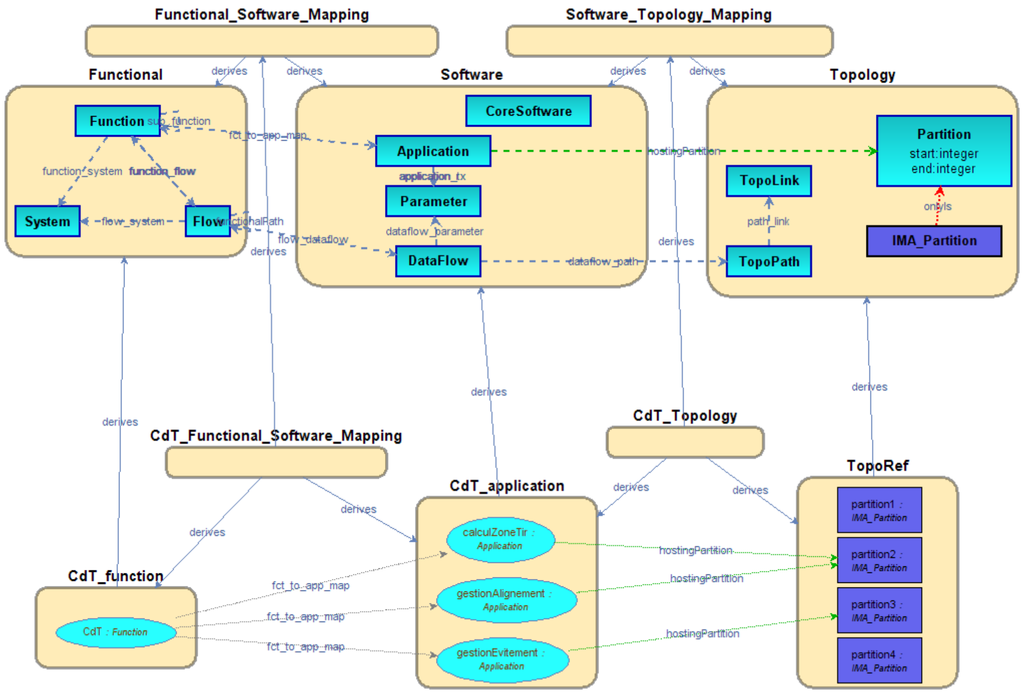


Figure 2: Worlds describing a product derives from generic ones

The same applies to constraints, the following requirement that every application is hosted by a partition is expressed in an intermediate SoftTopoRequirements world as this:

[3]Though this is not used in the context described in this article, the low level language is then compatible with SAT solvers tools that can be used to answer different questions.

```
world SoftTopoRequirements
     derives  Requirements,Software_Topology_Mapping  {
assert all_apps_allocated =
     forall entity a | a::Application =>
       exists entity p | p::Partition and hostingPartition[a]===p
}
```

This world is then derived by *CdTRequirements* that will gather all requirements applicable to CdT. There it is evaluated to *satisfied* by the prototype tool (this is why it is represented in green in the screenshot of Figure 3

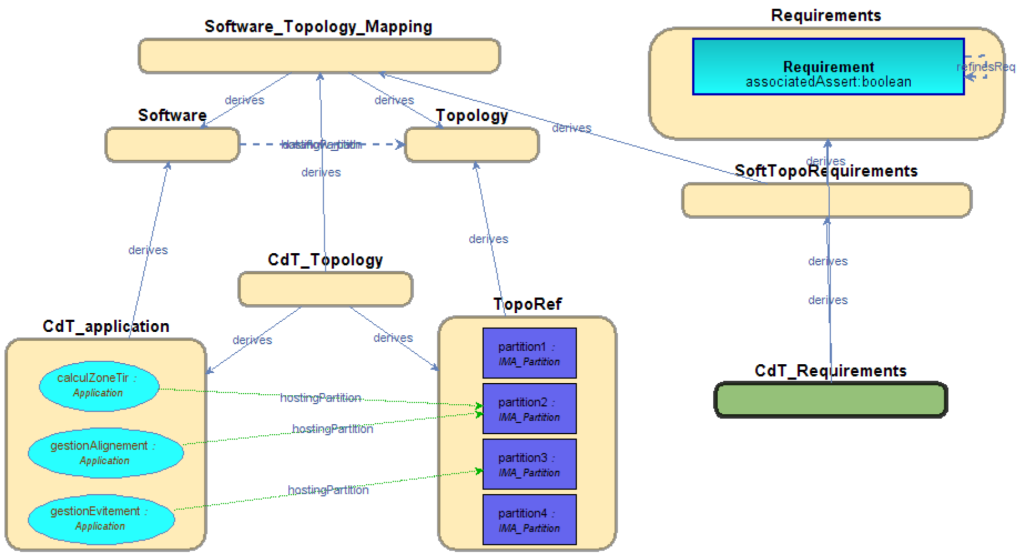


Figure 3: CdT requirements are gathered in the CdTRequirements world

3 Argumentation principles

When inquiring the certification of a system, the inquirer must provide a certification file. According to the Ministry of Defence, this should be “A reasoned, auditable argument created to support the contention that a defined system will satisfy the R&M requirements”[3]. The exact nature of the elements is not detailed but graphical representation or models are more and more part of this certification file. For the safety aspects, the Ministry of Defence even explicitly requires safety cases[4] from which our work is inspired. Anyway the provided elements tend to grow while the structure, in particular the precise intent of an element or another, is sometimes missing. At the same time, IMA (Integrated Modular Avionics) is becoming the standard while its certification offers some additional issues [6].

The goal of the MIMOSA Argumentation facet is to represent graphically the different means of compliance used to justify the satisfaction of the requirements. The Argumentation facet organizes the various elements (formal and informal) that contribute to the justification of requirements.

When coming to represent graphically argumentation, GSN [2] is a reference and MIMOSA argumentation strongly inspires from it. It also inspires from Toulmin works for underlying principles [7] and from existing work on assurance cases in the aeronautical domain [1]. In MIMOSA a generic argumentation step relies on the following concepts:

Claim the property to be justified (will often link to a requirement),

Evidence the facts that will be used to justify the claim (analysis results, test results, expert knowledge, bibliographical reference. . .),

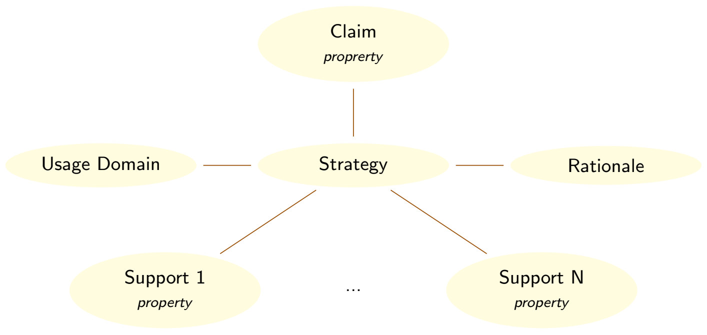


Figure 4: The generic argumentation step

Strategy combination of different evidences in order to justify a claim, is the model counterpart of "Mean of Compliance".

Usage Domain domain on which the method is usable

Rationale justifies the use of the method in this particular context

Argumentation patterns are then proposed as a partial instance of this generic step, specifying subconcepts, known entities, and their necessity status depending on the strategy or strategy family.

Argumentation patterns can be of different natures: generic (as "Using a software" that will make mandatory to explicit the usage domain and add a corresponding support to show its respect) or domain specific (like "Showing that the Worst Case Response Time of an application is bounded by a value").

When building an argumentation two mechanisms are then used:

- argumentation step chaining – one claim of a level becomes a support for the next one ;
- argumentation pattern composition – a given step inherits from more than one argumentation patterns.

In this last case, it has to exhibit the union of supports, usage domain and rationale of its "parents" (see Figure 5). To make sense strategy and claim of the parents should be compatible, for example "using a software to compute a WCRT bound" would inherit from the patterns "using a software" and "calculating a WCRT bound". See the example just below.

Note that the particular usage that is envisaged here protects us from what could have been an important issue: the supports consistency. In a general case, both mechanisms of combination of argument steps proposed just above would have, as an additional task, to prove that the resulting support set is consistent (in fact, this should also be explored for each elementary step). Each support is here considered valid (and the real situation consistent), that means that:

- an elementary argumentation steps having inconsistent support requirements is unusable (all its supports can not be fulfilled at the same time)
- when combining two steps for which supports have been provided, no inconsistency may appear.

The first item is one reason – the formal one – why argumentation step merging may not make sense [4]. For example if two argumentation patterns have been written to deal with two different situations, merging them is useless as no real case would match both situations and then it would be impossible to fulfill all the supports of the resulting pattern.

Eventually, a global constraint should be that every Requirement written –the system must have property P– has a corresponding argumentation claim –The system has the property P–

A certification argumentation is complete when every requirement has got a corresponding assert with proper method, supports and some times rationale and usage domain. This can

[4]The other reason why a argumentation step merging may not make sense is that it does not make sense from a business point of view.

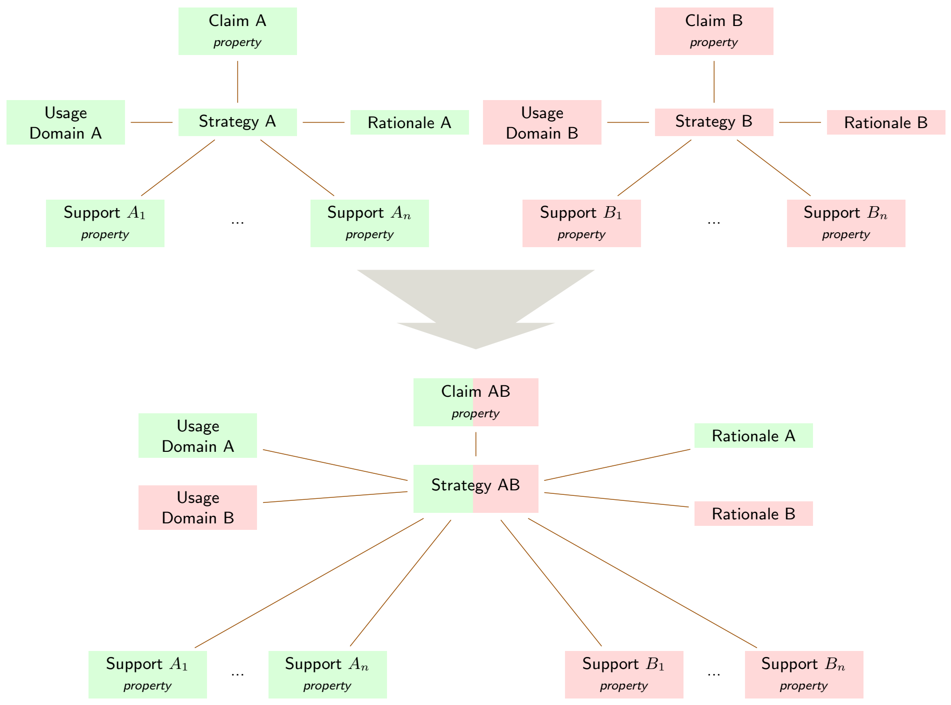


Figure 5: The argumentation step composition

be automatically checked by a tool then guiding the certification authority to missing asserts or argumentation steps.

4 Argumentation examples

"Using a software" is a generic pattern that mainly adds the constraint of exhibiting the *Usage Domain* (that is optional by default). This constraint activates another one that comes from the generic argumentation step telling that if a *Usage Domain* is attached to the *Strategy* then at least one support must show conformance to this usage domain.

The second argumentation pattern introduced is domain specific: "assessing an application maximum Worst Case Response Time". Such a claim needs that this WCRT is calculated, and relies necessarily on the computation of the Worst Case Execution Time of each involved applications in the partition hosting the evaluated one, and that the preemption policy and the period are known (links to architecture and realtime facet products will help precisely determine the relevant set of applications, as well as, if filled, automatically check preemption policy and period). A graphical representation of this pattern can be found in Figure 7. A constraint is added to ensure that, at least, WCET, preemption and period supports are provided if this strategy is used.

The third argumentation pattern is the composition of the two previous ones: "Using a software to assess an application maximum Worst Case Response Time" (see Figure 8). It then inherits from all the supports from the two previous ones, as well as the *Usage Domain* mandatory constraint coming from "Using a software".

5 Implementation and usage

Today, qualification/certification of new systems becomes more problematic due to several causes, the more invoked being the system complexity increase that is observed and a more practical one being the size of certification teams.

A less trivial reason is the evolution of the way system are developed: the development of a system is no more made from scratch with every subsystem developments made in consis-

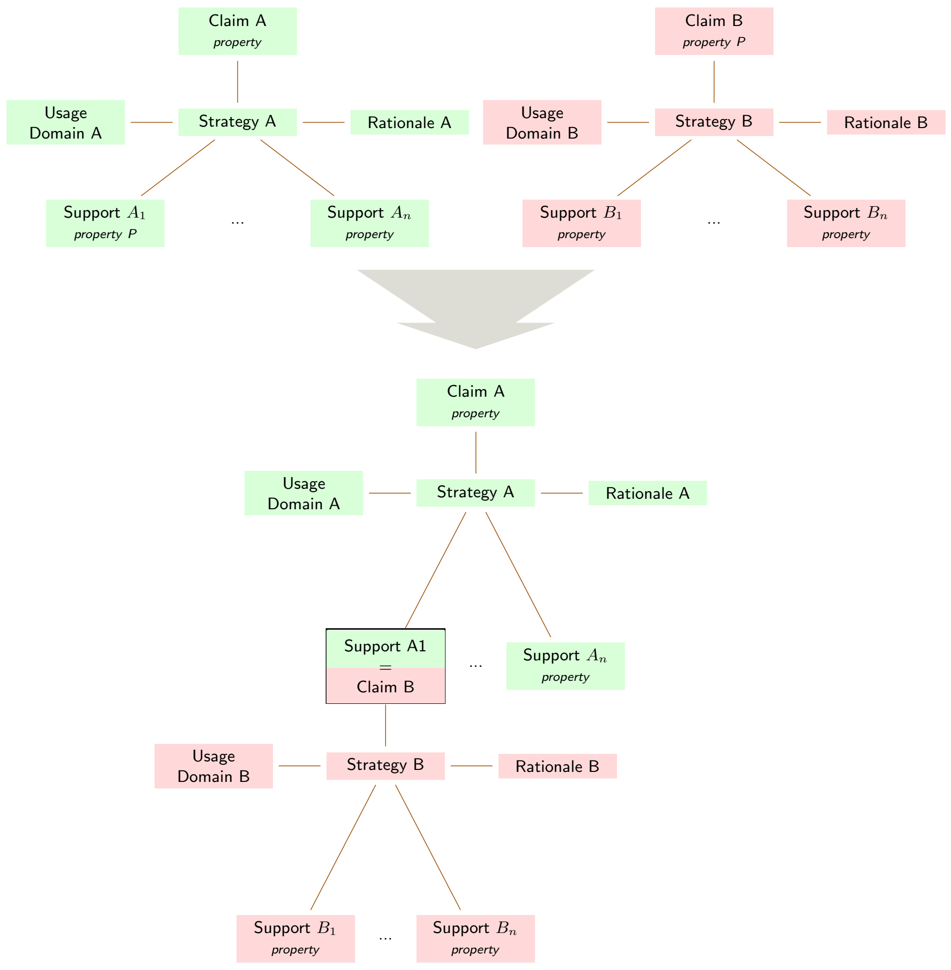


Figure 6: The argumentation step chaining

tency with the unique goal of integrating the system under development. On the opposite, a component based approach is more and more intensively used with a lot of reuse and "generic" components integration. This is strongly involved in the loss of structure of the certification file because part of the argumentation is then made either in another context (reuse case) or still at a general level (generic components case). Documentation provided to certification team is then not so organized, the same information may appear in many documents (strongly redundant documents may be provided to support the same kind of property but for different parts of the system due to the reuse of certification supports in another context and/or the pre-qualification steps). The certification file volume then grows up and, on the opposite of very redundant information, some precise information may become hard to find.

Formalizing the "structure" of the argumentation is the answer we promote in this work, by relying on the introduced argumentation pattern and combination mechanisms.

Apart from the formal principles enunciated in the previous section, the way one may use these argumentation patterns typically varies with the level of abstraction. Technical, low abstraction level patterns may benefit from formal inputs that a tool is able to manage, and provers or solvers can make automatic or semi-automatic reasoning from these argumentation steps. On the other extremity, high abstract level patterns are more likely to require human understanding and *supports* will often be links to heterogeneous documentation elements. In that last case, argumentation patterns must be seen as "check lists" and tooling will mainly guide the user to missing information, when some supports, rationale or usage domain are linked to no elements.

Classical envisage process of argumentation pattern is that the certification authority

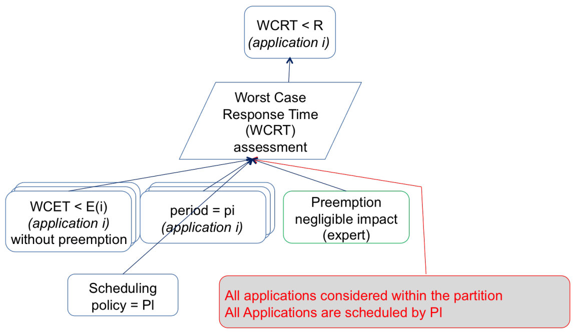

WCRT < R
(application i)
Worst Case Response Time (WCRT) assessment
WCET < E(i) (application i) without preemption
period = pi (application i)
Preemption negligible impact (expert)
Scheduling policy = PI
All applications considered within the partition
All Applications are scheduled by PI

Figure 7: bounded WCRT argumentation pattern

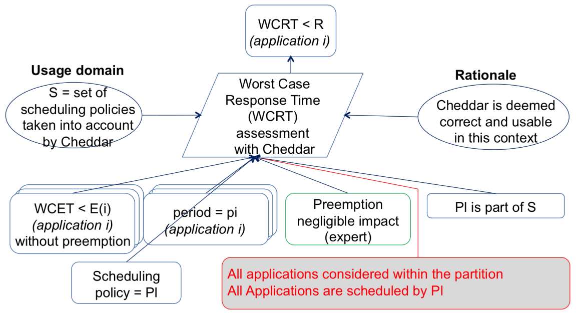

WCRT < R
(application i)
Usage domain
S = set of scheduling policies taken into account by Cheddar
Worst Case Response Time (WCRT) assessment with Cheddar
Rationale
Cheddar is deemed correct and usable in this context
WCET < E(i) (application i) without preemption
period = pi (application i)
Preemption negligible impact (expert)
PI is part of S
Scheduling policy = PI
All applications considered within the partition
All Applications are scheduled by PI

Figure 8: bounded WCRT by software computation pattern

shares validated patterns with industrial entity willing to obtain certification of a product. These patterns will act as guides for building a certification request for the industrial entity while supporting the analysis work of the certification entity, then being profitable for both. Capitalization is also made easier: certification authority builds a new pattern from analyzing a certification request that is not yet formalized this way, and then after an internal validation phase, add this new pattern as a new approved means of compliance. Industrial entity already having strong background in certification may also formalize their way of asserting a given property, and submit it to the certification authority as a new “standard” means of compliance –i.e. a new argumentation pattern– (this formalizes capitalization of the Certification Review Item process often used today when new approaches or technologies are to be evaluated).

6 Conclusion and perspectives

A Model based framework for certification process has been presented that offers to describe and analyze in a common language the system description, requirements that are put on it, and argumentation that is delivered to show the system conformance to its requirements.

Some argumentation patterns have been produced as well as a composition scheme that offers to build an instance of argumentation tree by mainly composing building blocks from rather domain specific ones to very generic ones.

A prototype has been developed to illustrate this approach, it has been tested on a case study containing two functions Fire control and Terrain following (then decomposed in several applications) allocated to a reference architecture with IMA capabilities ; and DGA TA envisages to use the approach on a real case study based on their certification activities. Nevertheless the exact way the methodology presented in this paper should take place is not yet completely defined, a shared tool between certification experts and candidate to certification would clearly be a good support but might not be realistic for tomorrow, an internal tool for the certification team that offers to build the argumentation based on the supports provided and some question/answer with the certification candidate may already help the certification team to organize, keep focused on the objective, and gain confidence in the decision they will eventually make.

References

[1] C. Michael Holloway. Explicate '78: Uncovering the implicit assurance case in do-178c. In *23rd Safety-Critical Systems Club (SCSC) Annual Symposium*, February 2015.

[2] Tim Kelly and Rob Weaver. The Goal Structuring Notation – A Safety Argument Notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[3] Ministry of Defence. *Defence Standard 00-42 Issue 2, Reliability and Maintainability (R&M) Assurance Guidance Part 3 R&M Case*, 2003.

[4] Ministry of Defence. *Defence Standard 00-56 Issue 4, Safety Management Requirements for Defence Systems Part 1 Requirements*, 2007.

[5] RTCA and EUROCAE. *DO-297/ED-124. Integrated modular avionics (IMA) development guidance and certification considerations.*, 2007.

[6] Jean-François Sicard, Ghilaine Martinez, and Florian Many. Specific Certification Issues Concerning IMA. In *Embedded Real-Time Software and Systems (ERTS2 2012)*, February 2012.

[7] Stephen E. Toulmin. *The Uses of Argument.* Cambridge University Press, Cambridge, UK, 2003. Updated Edition, first published in 1958.

Perspectives on Probabilistic Assessment of Systems and Software

Emmanuel Ledinot(1), Jean-Paul Blanquart(2), Jean Gassino(3), Bertrand Ricque(4), Philippe Baufreton(4), Jean-Louis Boulanger(5), Jean Louis Camus(6), Cyrille Comar(7), Hervé Delseny(8), Philippe Quéré(9)

(1): Contact author, Dassault Aviation, emmanuel.ledinot@dassault-aviation.com; (2): Airbus Defence and Space; (3): Institut de Radioprotection et de Sûreté Nucléaire; (4): Safran; (5): CERTIFER; (6): ANSYS-Esterel Technologies; (7): AdaCore; (8): Airbus; (9): Renault

Abstract:

Safety standards in most domains (aeronautics, automotive, industry, nuclear, railway, space) consider software (and more generally, design) as a deterministic artefact. They propose a global rationale combining probabilistic evidence on hardware random failures and deterministic evidence on systematic causes of failures including software. In a context where software is more and more pervasive in all systems, and where it is sometimes advocated that software complexity and size seem to provide some relevance to a probabilistic view of software behaviour, several initiatives suggest to change the way to address software in the global system safety assessment. This is a complex question with many facets. Among them the authors propose to discuss in the paper:

- foundations, relevance and limits of probabilistic assessment for software,
- relationship between software criticality category, (or class, DAL/SIL/ASIL/SSIL etc.) and probabilistic safety objectives,
- the rationale for software diversification and to what extent probabilistic assessment is part of it.

Keywords: software statistical testing, probabilistic system safety assessment, rationale of safety standards, DAL, SIL, ASIL, SSIL, cross-domain comparison.

1. Introduction – Position of the paper

Over the years, it has been recognized on the one hand that failures of safety systems may be due not only to random failures but also to "systematic causes" (be they called design faults, development errors etc.), and on the other hand that this kind of causes are less amenable to probabilistic assessment than e.g., random hardware failures. This is acknowledged and addressed by most existing safety standards under the form of the combination of quantitative (probabilistic) evidence and qualitative (deterministic) evidence, as appropriate regarding the various kinds of causes (see e.g., [Baufreton et al. 2010]).

The increasing role and complexity of software in systems, including safety critical ones, seem to make probabilistic evaluation of software reliability and statistical testing more and more attractive for some industry actors. They propose to at least modify the equilibrium between the various kinds of evidence in the global assessment framework (quantitative vs. qualitative, probabilistic vs. deterministic, etc.).

Originally motivated by technical discussions held within standardization committees, the work presented in this paper was undertaken by a French cross-domain group working on safety standards([1]). Software probabilistic assessment and statistical testing have been investigated for decades and a comprehensive bibliography would encompass hundreds of references. Starting from [Strigini et al. 1997], [Rushby et al. 2014] and [Ladkin et al. 2015], we address among the many facets of this complex matter the following aspects:

- whether and to which extent probabilistic assessment could be valid for software, particularly for safety critical systems,
- which relationship could exist between software criticality category (or class, DAL/SIL/ASIL/SSIL etc.) and probabilistic safety objectives,
- whether the benefits of solutions such as software diversification could be amenable to some quantification,

[1] Originally created in 2009 as part of the "Club des Grandes Enterprises de l'Embarqué" this working group on safety standards is now attached to Embedded France. It regularly publishes the results of exchange and common work between its members, experts in safety and related standards covering as many domains as aeronautics, automotive, industry, nuclear, railway, space. See e.g., [Baufreton et al. 2010], [Blanquart et al. 2012].

2. Technical Background

2.1. What is software statistical testing?

The characteristic feature of software statistical testing is probabilistic generation of test data. There are various purposes for doing so, and various ways of doing so. Following [Thevenod 91], [Thevenod 95], randomness on inputs may be used to *find faults* or to *assess dependability* at the end of the verification stage. In the former case, *coverage criteria* are the outcome of the statistical testing activity, as opposed to values of *probabilities* in the latter case.

The coverage criteria are based on activation counters logging how the items of the software are exercised by the generated input data. Depending on the nature of these items, statistical testing is qualified as *structural* or *functional*:

- structural if the items are implementation-oriented (e.g. control flow-graphs),
- functional if they are more specification-oriented like state-transition graphs or data-flow graphs.

When randomness is "blind", i.e. when the random input profiles are generated by means of uniform probability laws carrying no frequency information related to actual usage or operation, statistical testing is named *random testing*.

Random testing applies exclusively to statistical testing devoted to finding faults and fault removal. Random testing cannot be used for software dependability assessment. To support software dependability assessment, the randomly generated input data must be statistically consistent with the operational profiles.

It is sometimes claimed that statistical testing may support fault forecasting based on software reliability growth models, in addition to fault finding and removal.

Finding Faults		**Assessing Dependability**
Structural Coverage		Event Probabilities
Specification-oriented	Implementation-oriented	
Functional Statistical Testing	Structural Statistical Testing	Statistical Testing

Table 1: Types of statistical testing

In this paper we exclusively focus on statistical testing for software dependability assessment.

2.2. What is "software failure"?

A given piece of software, intended to perform a specified function, may be affected by faults in its functional requirements or by errors in the development process. When the inputs activate a fault, the computed outputs differ (deterministically) from the intended values and a system failure may occur.

While at random dates hardware components may *lose* some functional capability, software faults, when present, are present from the very beginning. The wording "software failure" inherited from hardware, electrical and mechanical engineering is convenient but misleading. When some definite inputs activate a fault into malfunctioning it may be *perceived* at system or user level as a random failure event. But "randomness" is present only in input and execution context variability, in other words in the operational profile (OP).

Whatever "software failure" event is quantified, a trigger of system safety, system reliability or system availability events, the software[2] to be assessed is a deterministic transformer of the inputs, which vary with some randomness according to the actual OP, hence the critical importance of OP modelling accuracy to ensure validity of the probabilistic assessment. This is the reason why we illustrate sensitivity to OP modelling by an example.

[2] Since our primary concern is safety critical software, software is assumed deterministic

2.3. Probabilistic modelling of software dependability assessment

A probabilistic model consists of a random experiment repeated with independence and observed by means of events that must meet Boolean algebraic properties. Then a probability measure is defined on these events [Rényi 2007].

In case of software statistical testing the random experiment is made of five steps, and is repeated N times where N is the sample size:

- *Set-up* of the program in a state common to the N experiments and ensuring *independence* between them. The more complex the software and its execution environment, the harder to meet this compelling requirement,

- *Random generation* of an input sequence (`IS`), consistent with OP probability law. OP denotes the set of all possible input sequences submitted to the program at operation-time, plus *frequency information*: a probability density function (pdf) defined over the set of all possible input sequences. In practice OP, as a probability law, is a very complex object: a high-dimension support set, plus a multivariate function defined on this set. The duration of one input sequence is that which is relevant for the quantified event: constant or variable, cycle, minutes, missions; hours, years etc. As previously mentioned, random generation does not mean random testing since the probability laws used on the program inputs are not all uniform. Some may be uniform, but this should not be a "by-default choice" because of lack of information but by explicit choice to ensure adequacy[3] with operational conditions.

- *Run* of the program with the generated `IS`,

- *Event evaluation* of the run. An event is a predicate defined on some observation variables common to all runs (program I/Os, internal variables, environment variables etc.). Depending on how far the specification is formalized and amenable to execution, the information gathering process to evaluate the event-predicate is automated or not. The event-predicate is the test oracle of deterministic testing,

- *Decision.* In the end, the random experiment is summed-up into a yes/no decision status on the event-predicate. This status is the run-specific realization of the Bernoulli random variable D associated to the event. It is a binary random variable. Its probability law is defined by:

    - o Pr(D=0)=p, where p is the parameter of the Bernoulli law of D. We assume that the event associated to D is stated so that "false" means "problem occurrence". Probability p is the risk of "software failure", which has to be estimated by means of the N-sample of independent runs.

    - o Pr(D=1)=q=(1-p) since the two events are exclusive and complementary. The realization of D at run n in N is like tossing a coin. Probability p is not necessarily very small, i.e. that of a rare event. It is the limit of C_0/N where C_0 is the number of (D=0) events in the N runs.

Once C_0 is known, it is not possible to compute p from N and C_0 only. One can choose p= C_0/N, which is the only sensible choice available. But another N-sample of runs would have given a different count C_0'. Hence the computed p would fluctuate if we repeated the building of samples made of N runs.

This problem is overcome by standard interval estimation of the parameter p for the binomial law of parameters N and k=C_0 associated to D. The binomial law is the probability law of the number of heads (resp. tails) observed after tossing N times the same coin of probability p. Handling the random fluctuations of C_0/N over repeated N-run-samples is analytically tractable.

Given an accepted risk of error on the computed p because of performing a *unique* N-run-experiment (this is a *design-office* risk, usually noted α, whose value is commonly *chosen* between 10% and 1%), one can compute an interval in which the true value of p is likely to lie. The probability of the design office event "the true value is *not* in the computed interval" is α. So the confidence on the interval is (1 – α). In the example of section 2.5 that illustrates the sensitivity to OP and the robustness issue, we took α **=** 1%.

[3] Validity or accuracy might be preferred as synonyms

2.4. Probability of fault freeness

Up to now, we did not address the quantity of faults in the software. We did not consider if there are any, nor how many they are and where they are. We restricted ourselves to an external view, just counting the discrepancies of the program's behaviour when observed through events and N runs.

Statistical testing is sometimes related to this second question of the quantity of residual faults in a piece of software. [Rushby et al. 2014] proposed a conceptual framework to encompass both issues:

- development assurance efficacy, that is estimating the likelihood of existence of residual faults in spite of a rigorous development, and possibly evolution of this likelihood over time (software reliability growth models),
- probabilistic software assessment, a snapshot view of software reliability, without the explicit estimation of the quantity of residual faults.

[Rushby et al. 2014] aims at bridging the gap between *deterministic software* correctness and *probabilistic system* safety. The authors attempt to define the probability of software perfection (fault freeness) and the probability of software failure under "randomly selected demand". They state the formula:

Pr(Sw Fails")=

Pr("Sw Fails" | "Sw is fault-free").Pr("Sw is fault-free") +

Pr("Sw Fails" | "Sw is not fault-free").Pr("Sw is not fault-free").

which we abbreviate in:

$$Pr(SF)=Pr(Sc).Pr(SF|Sc) + Pr(Snc).Pr(SF|Snc)$$

Consistently with the event evaluation stage in the random experiment definition (cf. .section 2.3), we interpret the notion of "Software Failure" in the following way: it assumes the existence of an oracle (computerized or human-based) over the observables of software Sw, and the existence of a set of selected runs that may violate this oracle. 'Sc' means 'S is correct (fault free)', and 'Snc', **n**on **c**orrect, is its negation. The formula is based on the total probability theorem for the "fault-free" vs "not fault-free" alternative.

We question the relevance of the concept underlying Pr(Sc) and Pr(Snc), in practice at the very least. In any case it is true that a fault-free software cannot activate a failure, so the conditional probability equation Pr(SF|Sc)=0 holds, and the formula simplifies to:

$$Pr(SF)=Pr(Snc).Pr(SF|Snc)$$

In [Rushby et al. 2014] different ways to estimate or upper-approximate Pr(Snc) are discussed, attempting to take into account the influence of development assurance levels (DAL, SIL, ASIL, SSIL). In 2.2 and 2.3 we tried to define Pr(SF|Snc) precisely and we sketched out how to compute its estimate using binomial parameter estimation.

We would like to underline that "randomly selected demand" has *no intrinsic meaning*: does it mean e.g., conformant to uniform laws on the inputs, to normal laws, to any arbitrary law on some physically significant combination of some inputs to be estimated in operation?

As already mentioned, Pr(SF) is critically dependent on OP, the probability law that drives the inputs to software [Strigini et al. 1997]. The Ariane 501 accident provided a spectacular example of the critical dependency of Pr(SF) on OP: a range change on a very single parameter (the horizontal velocity) and Pr(SF) jumped from ~0 to 1.

A more precise simplified formula making *explicit* the dependencies with respect to OP would be:

$$P_{OP}(SF)=P(Snc).P_{OP}(SF|\ Snc)$$

2.5. Robustness of software probabilistic assessment

As reviewed in [Ladkin 2015], there are many difficulties to overcome for functional statistical testing to be performed in a valid manner. We would like to underline an additional one: the possible instability, and hence absence of meaning, of the estimated probabilities with respect to great, or even tiny, variations on OP probability density function (pdf).

The example program is derived from the famous Bertrand's paradox [Rényi 2007]. It takes as inputs the coordinates of two points in the plane. It checks whether these points lie on the unit circle centered at the frame's origin and whether the length of the chord defined by the two points is greater than the side length of the encircled equilateral triangle. This geometric property is named (P). The chords meeting (P) are colored green, and the other ones are colored red.

```
function [status]=program(x1,y1,x2,y2)
epsilon=0.01;
R=1;
side=sqrt(3);

if abs(x1^2 + y1^2 - R^2) < epsilon &&
   abs(x2^2 + y2^2 - R^2) < epsilon,
   if sqrt((x2-x1)^2 + (y2-y1)^2) > side,
        status = 1;
   else status = 0; end;
else
    status = -1;
end
```

Figure 1: The source code of the program (distance computation and thresholding) derived from Bertrand's paradox

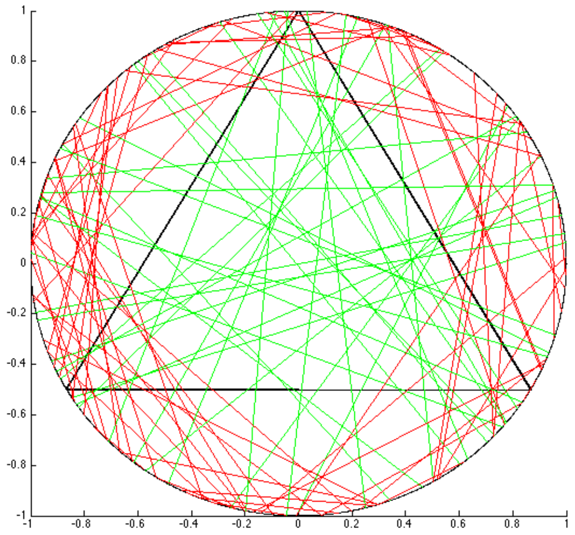


Figure 2: Functional statistical testing of (P) using uniformly generated random chords on the unit circle

We then use six different methods (D1 to D6) to generate the *uniformly* spread points over the circle (the ends of the chords). They only differ in the geometric construction of the points (Cartesian or polar coordinates etc.), all the random variables are uniformly distributed over their range ([-1,+1], [-π, + π], etc.). Two sample sizes are used (1 000 and 100 000). The results of the interval estimation of the binomial parameter at 99% confidence level for the statistical validity of (P) are the following:

	N = 100			**N= 100000**		
Sampling	**Pmin**	**Pe**	**Pmax**	**Pmin**	**Pe**	**Pmax**
D1	0,26	0,38	0,51	0,36	0,36	0,37
D2	0,22	0,34	0,47	0,33	0,33	0,34
D3	0,23	0,35	0,48	0,33	0,33	0,34
D4	0,27	0,39	0,52	0,33	0,33	0,34
D5	0,38	0,51	0,64	0,49	0,50	0,50
D6	0,38	0,51	0,64	0,50	0,50	0,50

*Table 2: the estimated probabilities of (P) for six different interpretations of "uniform" in the definition of the operational profile and two sample sizes. Pe is the **e**stimated probability p.*

The probability **e**stimate Pe ranges from 0.33 to 0.50, from 1/3 to 1/2, which is a surprising large variation when the six OPs that drive the sampling processes are expected to be similar enough to be considered *equivalent*. They are all eligible interpretations of "uniformly spread on the circle". Then, what is the meaning of these probabilities if they fluctuate so much for nearly undetectable reasons, uncontrollable in software verification practice?

2.6. Estimation of OP distribution laws

Example in section 2.5 illustrates the extreme sensitivity of software dependability assessment to the OP-pdf. Changes of OP-pdf have first order influence on $P_{OP}(SF| Snc)$ even for a program as trivial and regular as that of 2.5.

But even worse, when no change is intended on the law (let's say "uniform" as in 2.5), the way of building the test data generator conformant to this law leaves room to tiny degrees of implementation freedom that may have *also first order* influence on the estimation.

Unfortunately it is so whatever confidence level α is chosen. It is not a matter of estimator convergence and precision depending on N and α. Even with very small values of α (let's take 10^{-5}) and very large samples, the middle of the intervals can have chaotic jumps with respect to seemingly non significant changes in the implementation of the random generator.

This has consequences on qualification of COTS by proof in-use and service history. Extreme care must be taken as to the sensitivity of the computed probabilistic dependability indicators with respect to OP variability between the first and second usage context.

2.7. References to probabilistic software assessment in safety standards

Some standards such as EN 50128 refer to statistical testing as a possible means of software dependability assessment. Statistical evaluation is also referenced in an informative annex of part 7 of [IEC 61508] which states:

"*A probabilistic approach to determining software safety integrity for pre-developed software*". The text of this annex, now 17 years old, is very misleading. There are indications that very complex software such as operating system could be evaluated. This is not possible due to the very strict requirements applicable to operational history and data collection. These requirements are far beyond any practical application for such software. These critical aspects can remain unnoticed by a reader not deeply acquainted with the required mathematical statistics.

Furthermore, the state-of-the-art has significantly evolved since the references quoted in the standard. There is thus a clear need to reshape this text, clarify its mathematical foundations and define its possible scope of application.

Statistical testing is also mentioned in informative annex E of [IEC 60880]; anyway, this standard states that "*The validity of the calculated pfd depends upon the similarity of the profile of the test inputs to the profile of the actual inputs experienced by the system in operation. If (...) used on an unrealistic operational profile (...) a pfd will be estimated that may be very different to the actual system availability that would be obtained in active use. This is a fundamental weakness of the statistical testing approach as it is generally very difficult to accurately determine the operational profile that a system will experience in use, and this is particularly true for systems with large numbers of inputs.*"

The other standards considered in this paper (cf. References) do not resort to statistical assessment of software quality.

3. Software reliability and DAL/SIL/ASIL/SSIL

Given this setting, can we argue some link between development assurance levels and software reliability?

In system safety engineering, the probabilistic safety objective assigned to an entity determines its DAL by means of domain specific regulatory tables [Blanquart et al., 2012], [ED79A/ARP4754A], [EN 50129], [IEC 61508], [ISO 26262]. The converse is irrelevant, probabilities cannot be derived from DALs.

Since probabilities drive DALs at system level (resp. SIL, ASIL or SSIL in automation, automotive and railway), software items included, the question is "why not assigning DALs to legacy software or COTS by means of reliability measurement?" This would be some sort of reverse-engineered DAL, substantiated by product assessment instead of process assessment.

In process automation, some equipment and software vendors tend to lobby this way. In aeronautic, space, railway, and nuclear, rigour of component development is explicitly stated in standards as a *contextual* notion. It is system dependent, not specific to the component.

Component reuse or COTS use from system to system without dedicated component contextual evaluation, has to be performed with care. Masking system dependency, a reverse engineered DAL/SIL/ASIL/SSIL would be dangerously misleading.

4. N-version programming and system safety

4.1. Behavioral view .vs. probabilistic view

In all industrial domains it happens that hardware components reliability is too low to meet the catastrophic event probability objectives with single channel architectures. Duplex or triplex architectures introduce redundancies, possibly with hardware dissimilarity, to favour failure independence and thereby meet the quantified rareness objectives as low as e.g., 10^{-9} per hour or 10^{-5} failure on demand.

Because of the influence in the software arena of these compelling architectural patterns, or may be for the sake of uniformity in probabilistic safety and reliability assessment, there is some inclination towards quantifying “software failures”, and managing their statistical independence.

In addition, since DAL/SIL/ASIL etc. may be seen as process-based means to ensure quantified safety objectives over all items, including software, there are candidate probabilities for software failures at hand: that of the regulatory tables such as 10^{-9}/h for ASIL D in automotive, or 10^{-5}/h for DAL C in aeronautics etc. As mentioned in §3, the DAL/SIL/ASILs ensue from or may be linked to probability objectives.

Assuming that system and software development assurance meet their objectives, the probability of all “failures” of software S could be uniformly upper-bounded by the regulatory value associated to its DAL/SIL/ASIL.

But even then, even with these sensible but disputable probabilities for ‘software failures’, software dissimilarity would not be motivated by search for partial or complete *statistical independence*, at least for safety critical software[4].

For safety critical software, dissimilarity, also named N-versions programming, aims at *architectural* and *behavioural* objectives. The aim is to avoid single-cause catastrophic failures initiated by software (or double-cause in space domain). Dissimilarity copes with *possibility* of software-initiated catastrophic behaviours, not *quantity* thereof. It resorts to common cause analysis, for catastrophic effects caused by single (or double) residual faults in specification or implementation.

A 1-version piece of software mapped on k hardware redundancies may generate common cause failures in two ways:

1. On its own, when its k-replicated behaviour turns out to be catastrophic at system level,

2. As a *coupling influence* over the k hardware replicates, which may no longer be independent initiators (both *causally* and *statistically*), in spite of their possibly independent constituencies.

The first one is addressed by software development assurance, which encompasses formal methods to tend to elimination of correctness faults (conformance defects in the wording of standards).

The second kind is addressed by a set of best engineering practices to *isolate* the software behaviour from the execution platform, i.e. from any other influence than its specified inputs, initial state, and configuration parameters. Isolation best practices are detailed in the next section.

So in the setting of safety critical systems, N-version programming is advocated, possibly imposed, either because of confusion with hardware and plant architecting, or because of silent doubt on process assurance’s efficacy for the highest criticality levels.

4.2. Conditions for effective 1-SW k-HW redundancies

We consider a unique piece of software replicated on k hardware redundancies, which may be dissimilar or not. Are there conditions to ensure independence of hardware failures in spite of the potential common cause failure created by software uniqueness?

[4] Dealing with lower system/software criticalities (e.g. maintenance functions and system reliability).is another issue outside the scope of this paper.

Two-way isolation of the application software from its environment meets these conditions:

- No influence of the environment (operating system, hardware/software integration, etc.) on the functional behavior of the application layer. It must depend exclusively on its specified inputs and initial state.
- Conversely, no influence of the application software on the operating system, the execution platform, and more generally any external item other than the specified outputs.

4.3. Conditions for effective k-SW k-HW redundancies

In case of k-version programming over the k hardware redundancies, one may distinguish two situations:

- k-version at implementation level only,
- k-version at specification level and implementation level.

The intended meaning of specification here includes system requirements allocated to software, software high level requirements, functional requirements, design and low level requirements.

Considering k-version implementation to enforce software failure independence, [Knight et al. 1986] provided experimental evidence of non-effectiveness. Moreover, as software development methods significantly improved since the late 80s (model-based design, automatic code generation, model-checking), correctness of implementation is not the major concern.

In spite of software engineering progress, validity and completeness of system and software specifications remain a major issue. When possible, diversification at specification level would be beneficial. But it is most often than not very difficult to state the same algorithmic problem in two actually different manners, and then prove that the two formulations define the same set of expected behaviors…

Functional diversity (provision of different functions, e.g. based on different physical phenomena, to achieve the same safety objective) is even stronger than specification diversity, and is used in nuclear, space and other domains.

5. Conclusion

We tried to delineate some border lines in system and software safety assessment, mainly deterministic vs. random, behavioural vs. statistical.

These border lines are here and there left implicit in the standards, possibly because of some subtleties in their rationale, possibly also because of the limits of current engineering methods and tools.

We mainly focused on the validity conditions of probabilistic “software failure” estimation and on two system level aspects of probabilistic software assessment: design assurance levels and n-version programming. The validity of the operational profile distribution law and the sensitivity, possibly chaotic to this law, seemed to us the main impediments to probabilistic assessment of software dependability (not even mentioning the well-known difficulties related to the needed computation effort and time for ultrahigh reliability software, definitely an important issue as well though not addressed here).

Unfortunately, these impediments are even more hindering as software complexity increases, whereas statistical testing is sometimes advocated as an opportunity for greater cost effectiveness on very large software.

Driven by the increasing complexity of software and the trend toward ubiquitous systems of systems, there is some incentive to grant credit to statistical software assessment in safety standards under revision.

We explained why we remain extremely cautious about the validity of computed probabilities related to “software failures” and why we feel some danger in such a trend.

Basically, we reject, for ultrahigh reliability software, a move towards more statistical assessment against less development assurance. However, such a move may be debatable on low reliability software.

6. References

[Baufreton et al., 2010] P. Baufreton, JP. Blanquart, JL. Boulanger, H. Delseny, JC. Derrien, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, J. Machrouh, P. Quéré, B. Ricque, “Multi-domain comparison of safety standards”, ERTS-2010, 19-21 May 2010, Toulouse, France.

[Blanquart et al., 2012] JP. Blanquart, JM. Astruc, P. Baufreton, JL. Boulanger, H. Delseny, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, J. Machrouh, P. Quéré, B. Ricque, “Criticality categories across safety standards in different domains”, ERTS-2012, 1-3 February 2012, Toulouse, France.

[Butler et al. 1993] R. W. Butler, G. B. Finelli “The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software” IEEE Transactions on Software Engineering Vol 19 n°1 January 1993.

[ECSS-Q40] “Space product assurance – Safety”, European Cooperation for Space Standardisation, ECSS-Q-ST-40C, 6/3/2009.

[ED79A/ARP4754A] “Guidelines for Development of Civil Aircraft and Systems”, EUROCAE ED-79A and SAE ARP 4754A, 21/12/2010.

[EN 50129] “Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling”, CENELEC, EN 50129:2003, 7/5/2003.

[IEC 60880] “Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions”, IEC 60880, edition 2.0, 2006-05.

[IEC 61508] “Functional safety of electrical/electronic/ programmable electronic safety-related systems
IEC 61508 Parts 1-7, Edition 2.0, 4/2010.

[ISO 26262] “Road vehicles – Functional safety” ISO 26262 Parts 1-9, first edition, 2011-11-15, ISO 26262 Part 10, 2012-08-01

[Knight et al. 1986], J. Knight, N. Leveson, “An experimental evaluation of the assumptions of independence in multiversion programming,” IEEE Transactions on Software Engineering, vol. SE-12, pp. 96–109, Jan. 1986.

[Ladkin et al. 2015] Peter Ladkin, Bev Littlewood “Practical Statistical Evaluation of Software” Version 4 of 01/03/2015 – IEC 61508 – 7 Annex D Working Group.

[Rényi A. 2007] Probability Theory, Dover 2007.

[Rushby et al. 2014] John Rushby, Bev Littlewood, Lorenzo Strigini “Evaluating the Assessment of Software Fault-Freeness” AESSCS Workshop 13 May 2014 Newcastle upon Tyne UK.

[Strigini et al. 1997] Strigini L., Littlewood B. “Guidelines for Statistical Testing” - ESA/ESTEC Study Contract PASCON/WO6-CCN2/TN12 n°10662/93/NL/NB WO6-CCN.

[Thévenod 91] Pascale Thévenod-Fosse, Hélène Waeselynck “An investigation of statistical software testing”, Journal of Software Testing, Verification and Reliability, vol. 1, (2): 5--25, 1991.

[Thévenod 95] Pascale Thévenod-Fosse, Hélène Waeselynck, Yves Crouzet “Software statistical testing”. In Brian Randell, Jean-Claude Laprie, Hermann Kopetz, Bev Littlewood (eds.): Predictably Dependable Computing Systems, Springer Verlag, ESPRIT Basic research Series, pp. 253-272, 1995.

7. Glossary

ASIL	Automotive Safety Integrity Level
COTS	Commercial Off-The-Shelf (component)
DAL	Development Assurance Level
OP	Operational Profile
PDF	Probability Density Function
PFD	Probability of Failure on Demand
SIL	Safety Integrity Level
SSIL	Software Safety Integrity Level

Session 6

Network

Wednesday 27th, 14:45 – Guillaumet

AeroRing: Avionics Full Duplex Ethernet Ring with High Availability and QoS Management

A. AMARI [1], A. MIFDAOUI[1], F. FRANCES[1], J. LACAN[1], D. RAMBAUD[2], L. URBAIN[3]
[1]University of Toulouse-ISAE, France, [2]BetaTECH, France, [3]ECA Group, France,

Abstract—**The avionics standard AFDX has been introduced to provide high speed communication for new generation aircraft. However, this switched network is deployed in a full redundant way, which leads to significant quantities of wires. To overcome this limitation, a new avionic communication network, called AeroRing, is proposed in this paper to decrease the wiring weight, while guaranteeing the required performance and safety levels. AeroRing is based on a Gigabit Ethernet technology and implements a daisy-chain wiring scheme on a Full Duplex ring topology. First, the main features of such a proposal, and particularly the QoS and robustness management, are detailed. Then, numerical results of some Performance Indicators (PI) are illustrated to highlight its ability to guarantee the avionics requirements.**

Keywords-**Avionics, Real-Time Ethernet, Ring topology, QoS, Performance, Safety.**

I. Introduction

The inherent complexity and bandwidth requirement of avionics communication architectures are increasing due to the growing number of interconnected end-systems and the expansion of exchanged data. The Avionics Full Duplex Switched Ethernet (AFDX) [1] has been introduced to provide high speed communication (100Mbps) for new generation aircraft. However, this switched network is deployed in a full redundant way, which leads to significant quantities of wires, and thus increases weight and integration costs. For instance, the A380 contains 500 km of cables [2].

To cope with these emerging issues, an avionics implementation reducing wires will clearly improve the efficiency and reliability of aircraft through decreasing the integration complexity, and reducing fuel consumption and maintenance costs. Therefore, a new avionic communication network, called AeroRing[1], based on a Gigabit Ethernet technology and implementing a daisy-chain wiring scheme on a Full Duplex ring topology, is proposed in this paper to enhance performance, while guaranteeing a high safety level for avionics applications.

Nowadays, Ethernet technology is considered as one of the most cost effective solutions allowing scalable and arbitrary topologies, and supporting high speed communication and Quality of Service (QoS) requirements. Various approaches have been proposed to guarantee real-time communications on top of Ethernet. The most relevant Real Time Ethernet (RTE) profiles, supporting ring topology and cited in IEC 61784-2 [3], have been described in [4]. However, most of these existing solutions are optimized for specific use cases and present some limitations in terms of:

- *resource utilization efficiency*, since concurrent access to the medium is generally not allowed due to their implemented control mechanisms, e.g., master/slaves or TDMA;
- *flexibility*, as only pre-planned cyclic communications are enabled, and require an efficient synchronization protocol;
- *robustness management*, because of the centralized fault management, classically provided by the master or network manager, which is considered as a central point of failure.

Hence, the main contribution of this work is the specifications of a new RTE protocol, AeroRing, which bridges the gap between existing RTE solutions, supporting ring topology, to enhance the resource utilization efficiency, the system flexibility and robustness management; in addition to guaranteeing the key requirements of safety-critical domains, such as avionics.

In the next section, we review the most relevant RTE solutions supporting ring topology and relate them to our proposal. Afterwards, the main innovative features of AeroRing, and its basic functionalities including the QoS and robustness management, are detailed in Sections 3 and 4, respectively. Finally, in Section 5, some numerical results on performance and reliability indicators of such a proposal are presented, to highlight its ability vs avionics requirements.

II. Related Work

During the last two decades, a wide range of RTE solutions have been proposed by industrials and academia. The most relevant ones have been cited in the document IEC 61784-2 [3], which in addition introduces a set of Performance Indicators (PIs) to evaluate the RTE networks abilities. In this section, we first describe the most effective PIs and the main requirements to fulfill for safety-critical applications. Then, based on these requirements, we conduct a benchmarking of AeroRing and the most relevant IEC profiles supporting ring

[1]Aeroring is co-funded by the European Union. Europe is involved in Midi-Pyrenees through the European Funds for Regional Development.

Protocols	Costs	Reliability	Availability	Performance	Complexity	Ethernet Compatibility
EtherCAT	High	Medium	High	Very High	High	No
PROFINET IRT	High	Medium	High	High	High	No
Ethernet/IP with DLR	Medium	High	Medium	Medium	Low	Yes
AeroRing	High	High	Very High	High	Low	Yes

TABLE I
BENCHMARKING OF RTE SOLUTIONS SUPPORTING RING TOPOLOGY

Characteristic	EtherCAT	PROFINET IRT	EDLR	AERORING
Rate (Mbps)	100	100	100	1000
Topology	Bus or ring	Bus or ring	Daisy-chain ring	Daisy-chain ring
Media	100Base-TX	100Base-TX	100Base-TX	1000BASE-TX
Control Mechanism	Master/slaves	Master/slaves	DLR	Event-triggered with SP policy
Robustness management	centralized	centralized	centralized	distributed
QoS management	no	no	yes	yes
Standardization	Open standard	Open standard	By OADV	Open specifications
Pros	On-the-fly transmission Short transmission cycle	Cut-through transmission Short transmission cycle	Efficient faults detection QoS Management	Cut-through transmission Short transmission cycle QoS Management Distributed Fault Management
Cons	Specific devices Central point of failure	Specific devices Central point of failure	Complexity due to integrated switches High latency	Not standardized yet

TABLE II
SPECIFICATIONS COMPARISON OF RTE SOLUTIONS SUPPORTING RING TOPOLOGY

topology.

A. Performance Indicators and Requirements

Among the specified PIs in [3], we consider the following main ones:

- Maximum Delivery Time, indicating ”the time needed to convey an APDU containing data (message payload) that has to be delivered in real-time from one node (source) to another node (destination)” when considering the worst-case scenario.
- Fault Detection Time, indicating the maximum time needed to all nodes to be aware of failure.
- Redundancy recovery time, indicating ”the maximum time from failure to become fully operational again in case of a single permanent failure”.

Furthermore, we consider an additional PI, which is the maximum backlog to evaluate the memory utilization within network components. Numerical results concerning these aforementioned PIs of AeroRing are illustrated in Section V.

In addition to these PIs, RTE networks must fulfill a set of key requirements, which reveal particularly effective for many safety-critical applications, and particularly for avionics. These requirements concern both technical and costs aspects. The technical requirements are mainly the timeliness and the accuracy of delivered data, in addition to the reliability and availability of the communication network. Furthermore, the choice of the RTE solution shall be efficient to meet the design requirements for the least amount of money. Therefore, the IEEE802.3 compatibility, a minimized configuration effort and reduced implementation costs are among the most important issues to guarantee. These requirements will be considered to benchmark AeroRing against the most relevant IEC profiles, supporting ring topology and cited in IEC 61784-2 [3].

B. Benchmarking Most Relevant IEC profiles

Among the RTE solutions in [3], there is a first class with an implementation at the network layer, e.g., P-NET, V-NET, Modbus-RTPS and Ethernet/IP. These solutions are usually easier to implement and configure, but they lead at the same time to important latencies (about 10ms), which makes them more effective for soft real-time applications. Then, there is a second category providing a realization on top of the MAC layer while keeping the IEEE802.3 compatibility, e.g., TCNET, Ethernet/IP with Device Level Ring (DLR) and PowerLink, or modifying the standard implementation, e.g., EtherCAT and Profinet IRT. In this paper, we focus only on the most relevant RTE solutions supporting ring topology, and particularly EtherCAT, Profinet IRT and Ethernet/IP with DLR.

EtherCAT is defined by Beckhoff GmbH and supported by the EtherCAT Technology Group (ETG). It implements a master/ slave mechanism on top of Fast Ethernet (100Mbps). The main particularity of EtherCAT is the on-the-fly forwarding technique, which allows the slaves to insert the data requested by the master directly in the frame crossing couplers step by step. The EtherCAT frame is transmitted from the first slave to the last, and then back in the opposite direction to the master. This protocol provides interesting real-time performances due to the on-the-fly mechanism.

However, the main drawbacks of this technology consist of:

- the specificity of the EtherCAT devices, which increases

the implementation costs and the configuration efforts;
- a central point of failure (i.e. the master) decreasing the reliability level.

PROFINET IRT (*Isochronous Real-Time*) is an extended version of PROFINET, which supports real time communications. It is a master/ slave network, based on cyclic communication handling two communication channels: isochronous and asynchronous. These latter are used by slaves to transmit real-time and non real-time data, respectively. The data is relayed using ”Cut-through” to reduce the processing time. These functionalities require specific equipments and an accurate synchronization protocol. This protocol has similar pros and cons than EtherCAT due to its incompatibility with IEEE 802.3 and its master/slave mechanism.

Device Level Ring (DLR) protocol was introduced in 2008 by OADV organization to support hard real-time communication with Ethernet/IP. DLR is based on a ring controller, called active ring supervisor, which collects data from the other interconnected nodes on only one port to avoid infinite traffic loop, except some specific frames, i.e., beacons. Each equipment has two Ethernet interfaces and an integrated switch, which implements Store & Forward mechanism and Static Priority service policy. Moreover, fault detection and reconfiguration mechanisms are handled within the controller via specific messages, i.e., beacon and announce. This protocol has interesting features in terms of reliability due to the fault detection mechanism within the controller, and reduced costs due to standard devices. However, the non-nominal case needs the reconfiguration of the supervisor, which increases the configuration effort. Furthermore, integrated switches based on Store & Forward mechanism induce high transmission latencies, which decrease the offered real-time performance and availability levels.

The benchmarking of these RTE solutions vs the main identified requirements in Section II-A is illustrated in Table I. EtherCAT and Profinet IRT imply higher costs due to the specificity of the implemented devices, and lower reliability due to the master/slaves mechanism (i.e. inducing a central point of failure), than Ethernet/IP with DLR. This latter is based on standard devices and implements fault detection and reconfiguration mechanisms, which enhance costs and reliability. Concerning real-time performance, EtherCAT and Profinet IRT allow very short latencies due to on-the-fly and Cut Through mechanisms, whereas Ethernet/IP with DLR induces high latencies because of the Store & forward one. Moreover, these transmission latencies have a direct effect on the fault detection time, and consequently the availability level. Hence, the offered real-time performance and availability levels of EtherCAT and Profinet IRT are higher than Ethernet/IP. It is worth noting that each RTE solution satisfies selected requirements better than others, but there is no best solution in terms of all the requirements.

Our objective is to specify a new RTE solution, AeroRing, which bridges the gap between these aforementioned RTE solutions, to guarantee the high reliability level of Ethernet/IP with DLR and the high real-time performance and availability levels of EtherCAT and Profinet IRT. Moreover, this new RTE solution has to keep the IEEE802.3 compatibility and reduce the implementation costs and configuration efforts. Hence, the AeroRing abilities vs the the main identified requirements are illustrated in Table I.

The main innovative features of AeroRing are as following:
- ***Distributed access mechanism***, allowing simultaneous data exchange to increase the offered bandwidth and resource usage efficiency;
- ***Distributed fault management mechanism***, avoiding the central point of failure to provide high *Reliability* and *Availability* levels;
- ***Event-triggered communication***, enhancing the system flexibility and decreasing the implementation complexity, through avoiding any need of synchronization;
- ***QoS management***, handling heterogeneous data constraints. This feature is guaranteed through the implementation of a Static Priority (SP) policy supporting four traffic classes: the network management class with the highest priority, the Hard Real Time (HRT) with the second highest priority, the Soft Real Time (SRT) class with medium priority and finally the Non Real Time (NRT) class with the lowest priority.
- ***QoS-aware routing algorithm***, sending HRT on both ports to improve reliability, and the SRT and NRT traffic classes on the shortest path to enhance resource usage efficiency;
- ***Compatibility with IEEE802.3***, guaranteeing an easy deployment process and a cost-effective integration.

Table II illustrates a summary of the main characteristics of AeroRing and the aforementioned RTE solutions, and particularly the pros and cons of each solution.

III. What is AeroRing

In this section, we present the main fundamental concepts of AeroRing network, including T-AeroRing features and data processing.

A. T-AeroRing Features

As illustrated in Fig. 1, AeroRing network implements a daisy-chain wiring scheme on top of a Full Duplex ring topology. It allows any ”Ethernet-compliant” equipment to transmit its data via a specific end-system, named T-AeroRing. Each transmitted packet will be forwarded from one T-AeroRing to another until reaching the final destination, and some particular cases will be detailed in the next section.

The T-AeroRing is a specific 3 ports Full Duplex Ethernet switch having the internal architecture illustrated in Figure 2, and the following main characteristics:

- ***Cut-Through forwarding technique***: the T-AeroRing starts forwarding the packet just after its identification, i.e. only the header of each packet is decoded to determine its destination port. This technique guarantees shorter transmission latency than the ”Store and Forward” technique (implemented within Ethernet/IP), which waits until the complete reception of the packet before forwarding it to the destination port;

- ***Static Priority service policy*** packets are queued in each output port of T-AeroRing according to their priorities. A queue is selected for transmission only if all traffic classes queues with higher priorities are empty. Then, for each queue, the scheduling order is First In First Out (FIFO) with a non-preemptive transmission. Priority is defined according to the IEEE 802.1p standard where the 802.1Q tag (3-bits field) is used to manipulate the four priority classes;

- ***Traffic policing***: To guarantee real-time performance, the T-AeroRing implements traffic policing mechanisms, based on Leaky Bucket method and particularly greedy method [5], to control each traffic class compliance with its predefined contract to avoid the network saturation. These traffic contracts are defined based on the network designer specifications. Each equipment connected to a T-AeroRing should be aware of these traffic contracts, and may apply traffic shaping to ensure the conformity of its generated traffic and avoid being discarded by the traffic policers. Each traffic exceeding its associated contract may be discarded to guarantee the communication determinism;

- ***QoS-aware routing***: unlike COTS Ethernet switches which relay frames on the basis of the address learning process and the Spanning Tree Algorithm, each T-AeroRing builds its routing table on the basis of the network management messages, exchanged between the interconnected T-AeroRings during the initialization phase or when a topology modification occurs (i.e. failure or restoration). Each T-AeroRing implements two routing modes to transmit its generated packets depending on their priorities: (i) sending on both ring ports (Ports 1 and 2 in Fig. 2) for high priority traffic classes, i.e., network management and HRT data, to allow a high reliability level ; (ii) sending on the port corresponding to the shortest path for medium and low priority traffic classes, i.e., SRT and NRT data, to offer a high performance level (i.e. short delay);

- ***Frame Redundancy Management***: Like AFDX end-systems, the T-AeroRing implements a Frame redundancy management mechanism to detect redundant frames generated by the first routing mode, and to determine whether to deliver the packet to the final destination or drop it because its replica has already been received. In practice, all packets sent on both ring ports are provided with a 2-bytes sequence number field that occurs just before the FCS field, which will be checked at the destination;

- ***Filtering Function***: To avoid infinite packet looping as a result of broadcast communication or erroneous header information, each T-AeroRing implements a filtering function which consists in: (i) eliminating all its generated packets sent on one port and received on the other port. This case occurs for broadcast packets or those with erroneous destination address; (ii) eliminating all received packets with erroneous source address. This verification is possible due to the cut-through technique and the routing table, i.e. an erroneous address does not exist in the routing table.

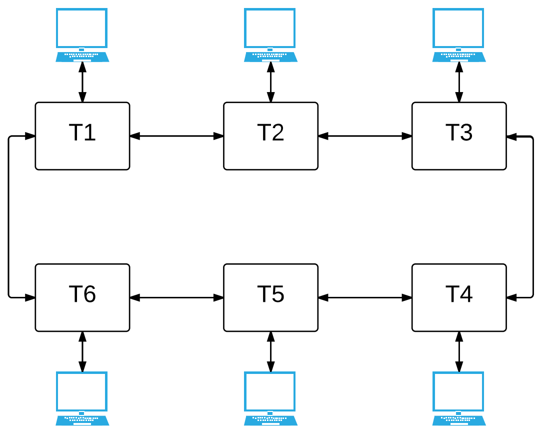


Fig. 1. AeroRing network

B. Data processing

Based on the description of T-AeroRing ports in Fig. 2, each frame will be processed as follows in the nominal case:

- Any frame received on a network port (1 or 2) is relayed to the other port unless the frame is destined to the connected equipment, or this latter is the frame source.
- Any frame received on a network port (1 or 2) destined to the connected equipment is delivered to it, according to the redundancy management mechanisms.
- Any frame received from the connected equipment is transmitted on one or both network ports depending on its priority: the highest priority traffic class is tagged by a 2-bytes sequence number and transmitted on both ports, while the medium and lowest priorities traffic classes are transmitted on the port corresponding to the shortest path.

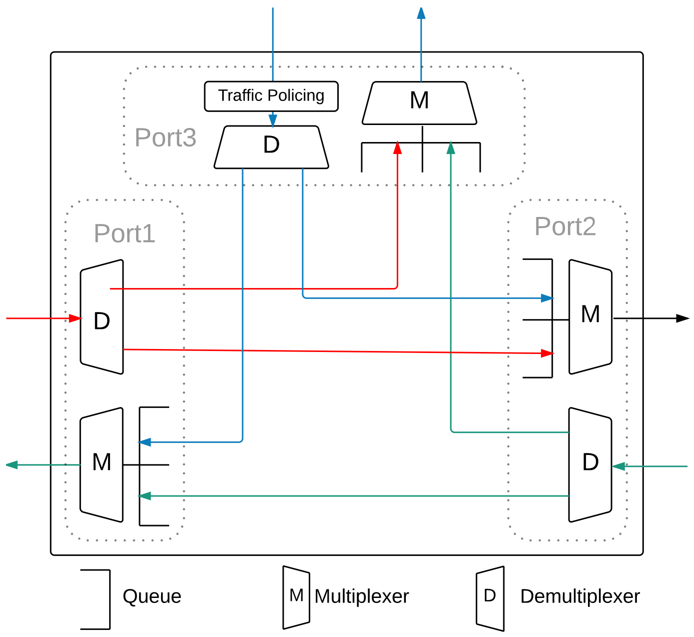


Fig. 2. *T-AeroRing* internal architecture

IV. Network Functionalities

In this section, we present the main functionalities of AeroRing, including the QoS and robustness management and the auto-configuration mechanisms.

A. Real-Time behavior and QoS Management

The real-time behavior of AeroRing and the timeliness guarantee of the delivered data are favored due to the implemented features within the T-AeroRing. First, the ”Cut Through” forwarding technique allows a short transmission time along the network, which improves the Maximum end-to-end delivery time. Then, the traffic policing mechanism prevents a network saturation by a deficient equipment, which guarantees the communication determinism. Furthermore, the implemented QoS-aware routing algorithm supports the transmission of the SRT and NRT data on the shortest path, which decreases their transmission delays. Finally, the Static Priority policy ensures the temporal isolation between mixed criticality data with various temporal constraints, and guarantees a bounded delay for the HRT traffic class.

On the other hand, AeroRing guarantees QoS management through the implementation of ”Static Priority” policy, which supports the following traffic classes:

1) ***Control Messages***: This traffic class has the highest priority level (N0) and is used for network management issues, such as: (i) building the routing tables of the interconnected T-AeroRings; (ii) the fault detection management; (iii) neighbor status checking. For the two former, the control messages are sent on both ring ports to ensure a high reliability level; whereas for the latter, the control messages are only sent to the neighbors, if no data to send, to check their status and guarantee a fast fault detection;

2) ***HRT Messages***: this traffic class has the second highest priority level (N1) and is assigned to real-time applications with hard temporal constraints, i.e. the message must be received before its deadline otherwise it is considered lost. This type of messages is sent on both ring ports to ensure a high reliability level, and is identified by a 2-bytes sequence number, essential for the frame redundancy mechanism within the destination T-AeroRing;

3) ***SRT Messages***: this traffic class has the medium priority level (N2) and is assigned to soft real-time applications, such as audio or video transfers. This type of messages is sent on the ring port corresponding to the shortest path to guarantee a high performance level, i.e. short transmission delay;

4) ***NRT Messages***: this traffic class has the lowest priority level (N3) and is assigned to non real-time applications, such as file transfer. This type of messages is sent on the ring port corresponding to the shortest path to guarantee a high performance level;

It is worth noting that the AeroRing is compatible with the IEEE 802.3 standard and each *T-AeroRing* can deliver any type of ”802.1x-compliant” frame from the equipment. Hence, if the frame does not include the 802.1Q tag, then it will be treated as a NRT data frame (N3), and transmitted on the ring port corresponding to the shortest path.

B. Auto-Configuration Mechanism

To reduce the configuration effort for the network designer and facilitate this new RTE solution adoption in the market, AeroRing offers an auto-configuration service until all the T-AeroRings become operational. This service is based on a simple address assignment method and a dynamic network topology discovery process. The address assignment of the connected T-AeroRings method consists in assigning the equipment MAC address to its corresponding T-AeroRing, when it joins the network. This fact facilitates the communication between the connected equipments and avoids a heavy translation addresses step.

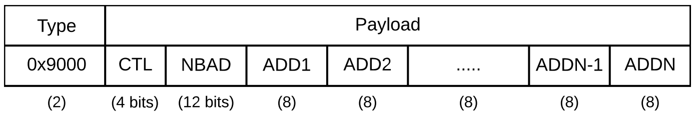
At the beginning, each T-AeroRing which is not connected to an equipment, has a default address MAC. Then, each freshly connected T-AeroRing enters a network topology discovery phase and behaves as follows:

- transmit periodically control messages on each ring port until receiving a control message from another T-AeroRing on the same port. This means that it has a neighbor on that side and it is no longer the last node of the segment. This period can be tuned according to the application requirements by the network designer.

- stop transmitting control messages when detecting both neighbors. Hence, the control messages transmission stops when the ring loop is closed.

Furthermore, on the basis of these control messages exchanged between T-AeroRings, each T-AeroRing builds a routing table per port. These routing tables allow to select the port corresponding to the shortest path (ports 1 or 2) for a destination. Each exchanged control message contains the list of MAC addresses of the crossed T-AeroRings, according to their physical positions along the network.

Figure 3 shows the structure of a control message. The control messages are identified by the value type of ”0x9000”. Then, the CTL field identifies the type of the control message, where ”0001” is reserved to build the routing tables, and NBAD field is a counter of the MAC addresses, which are inserted in the ADDx fields.



Type	Payload						
0x9000	CTL	NBAD	ADD1	ADD2	.....	ADDN-1	ADDN
(2)	(4 bits)	(12 bits)	(8)	(8)	(8)	(8)	(8)

Fig. 3. Structure of a control message

These control messages to build the routing tables are managed as following:

- at each topology change, i.e. start, failure or restoration, the T-AeroRings detecting this event send a control message on both network ports in broadcast mode with the highest priority, to update the routing tables of the other interconnected T-AeroRings;
- each T-AeroRing contributes in building the routing tables: when receiving the control message, it inserts its MAC address at the end of the list to respect the physical order, increment the NBAD counter, forwards it to the next T-AeroRing and updates its routing table (i.e. inserts MAC addresses of new equipments and deletes the ones that no longer exist) (see the example in Fig. 4).

C. Robustness Management

AeroRing offers a high reliability and availability levels due to its implemented features. First, redundancy management mechanisms are defined for both network and frames. Then, error detection and recovery mechanisms are implemented. Finally, a distributed fault detection and reconfiguration management is supported, which avoids single point of failure.

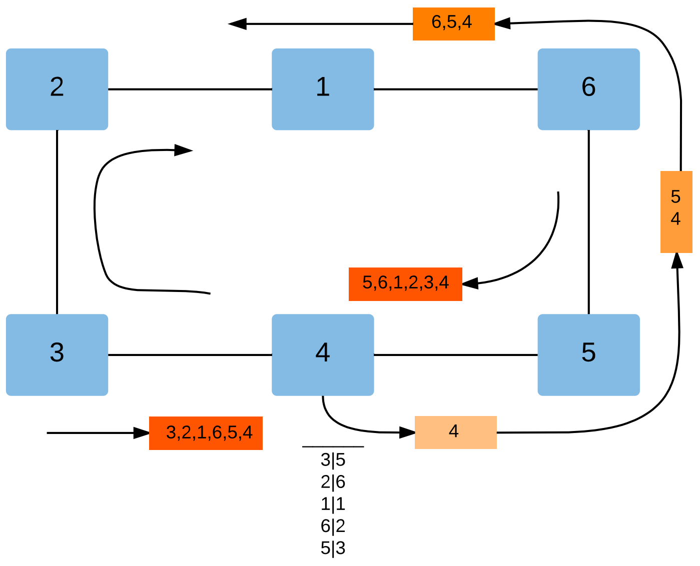


Fig. 4. Example of routing table building

1) Redundancy Management: AeroRing supports redundant topology where each equipment may be connected to redundant T-AeroRings and transmit its data on two redundant networks (see Fig. 5). These redundant networks might be used in a complementary way, i.e., the packet is sent only on the main network in the nominal case and on the backup one in case of failure; or in a symmetric way, i.e., each packet is replicated and sent on both networks. Each equipment implements a redundancy management mechanisms to identify packets (replicas) that arrive on both networks to consume the packet or drop it because its replica has already been received.

Furthermore, according to the QoS-aware routing algorithm within the T-AeroRing, the HRT traffic is sent on both ring ports to enhance the availability and reliability level. Each T-AeroRing implements a Frame redundancy management to detect redundant frames to determine whether to deliver the packet or drop it.

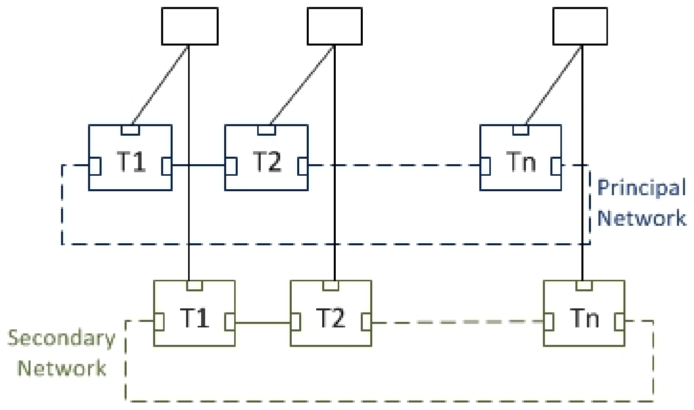


Fig. 5. Example of an AeroRing redundant topology

2) Error Detection and Recovery Mechanisms: Similar to standard Ethernet solution, AeroRing supports the error detection through the FCS field to discard erroneous frames. However, if the error is not detected based on the FCS field, and it occurs on the header, then the frame has to be eliminated from the network to avoid infinite packet looping. Each T-AeroRing implements a filtering function to prevent this phenomena. A frame with an erroneous destination address will be filtered by its source. However, if the source address is incorrect, any T-AeroRing can eliminate the erroneous frame when detecting that the erroneous address does not exist in its routing table.

3) Fault Detection and Reconfiguration Mechanisms: Any T-AeroRing has to consider a connection as down with a neighbor, if it does not receive any message from its neighbor during a certain period called ”detection period”. This detection period can be easily tuned by the network designer. In practice, if a T-AeroRing has no data to transmit to its neighbor, then it announces periodically its status to that neighbor through sending control messages. These latter have the CTL field set to ”0000”, and empty NBAD and ADDx fields (see Fig. 3).

These control messages to announce the status to neighbors are sent periodically when at least one of the following conditions is satisfied:

- The *T-AeroRing* does not have any data to send on this port during a period called ”*announcing period*” (this period is less then the *detection period* that covers in general the reception of more than one control message);
- The *T-AeroRing* did not receive any data or control message from this port for a duration equal to the *detection period*. In this case, the T-AeroRing indicates to its neighbor through a control message that the connection is considered as down.

When a connection is considered as down by one of the interconnected T-AeroRing, this latter sends a first control message to inform the other T-AeroRings with the CTL code ”0010”, followed by a second control message to update the routing tables (see the example in Fig. 6). A down connection is considered operational again (up), if the T-AeroRing starts receiving frames (data or control) from its neighbor. In this case, it sends a control message to update the routing tables of the other nodes.

It is worth noting the existence of various redundancy protocols for RTE solutions with ring topology, and the most relevant ones in our case are the Distributed Redundancy Protocol (DRP) [6] and the Ring-based Redundancy Protocol (RRP) [7].

The DRP implements similar local fault detection mechanisms than AeroRing, where each equipment can check the status of its neighbors by sending a link test frame

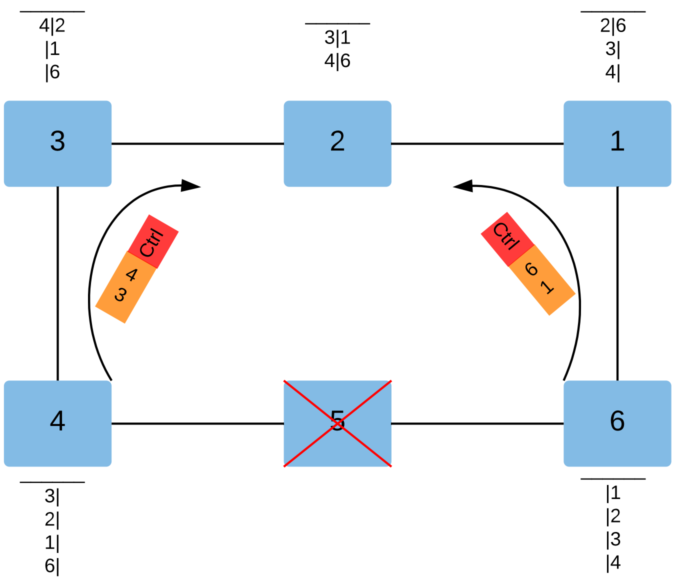


Fig. 6. Fault detection Mechanism

LinkCheck to detect failures. However, unlike AeroRing, in addition to these local mechanisms, DRP implements a centralized fault detection mechanism to check the ring status in a cyclic manner, i.e., during each cycle, only one equipment can check the ring status via a ring test frame *RingCheck*, gather and broadcast the information to the rest of equipments. Furthermore, an accurate synchronization protocol is required to manage such a cyclic process.

On the other hand, the RRP implements similar distributed mechanisms than AeroRing to build the routing tables within equipments. However, unlike AeroRing, RRP consists in transforming the ring topology into line topology to avoid infinite packet looping, through selecting two adjacent devices, called Ring Network Managers (RNMs), which disable one of their ports. This choice will clearly deteriorate the reliability level, compared to AeroRing, since sending on both directions becomes forbidden. Moreover, RRP implies higher communication overhead to build the routing tables than AeroRing, i.e., there are as many exchanged messages as equipments to update the routing tables under RRP, whereas only one control message is necessary with AeroRing.

Hence, unlike DRP, AeroRing implements a completely distributed redundancy protocol, based only on local fault detection mechanisms and without any need of synchronization protocol. Furthermore, unlike RRP, the T-AeroRings build autonomously their routing tables with a low induced overhead, and messages can be sent on both directions or only on the shortest path, according to the message class.

V. Performance and Reliability Analysis

In this section, we investigate the offered performance and reliability levels of AeroRing through a representative case study. Therefore, we give first numerical results on some specified PIs, such as the upper bounds on the end-to-end latencies and backlogs, and the maximum detection and recovery times. The detailed analyses of these PIs are beyond the scope of this paper.

A. Case study

We consider the following assumptions:

- The links speed is $C = 1Gbit/s$;
- The network size varies from 4 to 40 nodes (40 is the medium size specified in the IEC document [8] to benchmark RTE solutions);
- All equipments are similar and send the same traffic in broadcast mode;
- Technological latency within the *T-AeroRing* is $600ns$;
- Each equipment generates 3 types of traffic classes (TC) as described in Table III;
- The ”detection period” for fault management is 0.5ms.

It is worth noting that in the broadcast mode, the notion of ”shortest path” does not exist for the traffic SRT and NRT. In this case study, we consider that all the SRT and NRT messages are sent on the same direction, which corresponds to the worst-case scenario in terms of performance, i.e., this choice increases contention.

TABLE III
Traffic Characteristics

	TC	Payload (byte)	Throughput (Kbps)
I/O data	HRT	64	80
Audio streaming	SRT	100	128
File transfer	NRT	1000	500

B. Numerical Results

Figures 7 and 8 illustrate the upper bounds on the end-to-end latencies and backlogs, respectively. Obviously, these metrics increase with the network size, since the number of generated messages and crossed nodes increases.

As we can notice, for a network of 40 nodes, the upper bound on the end-to-end latency for the highest priority is less than 2ms, and the maximum backlog is less than 300Kbit.

Figures 9 and 10 illustrate the maximum detection and redundancy recovery times. These metrics depends on the length of the control message updating the routing tables, which increases with the number of crossed nodes. For a network of 40 nodes, the maximum detection and recovery times are less than 0.75ms and 1ms, respectively.

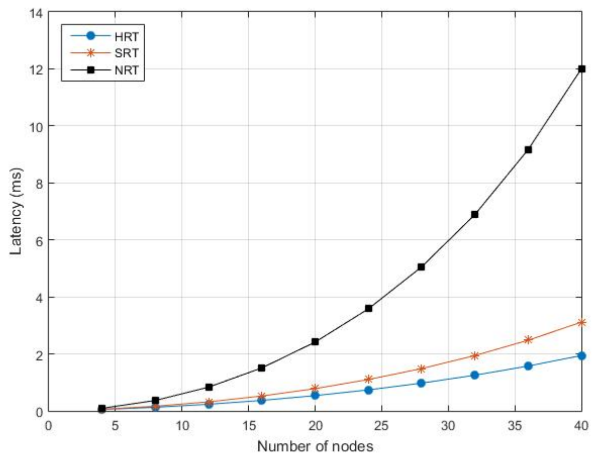


Fig. 7. Upper bounds on the end-to-end latencies vs number of nodes

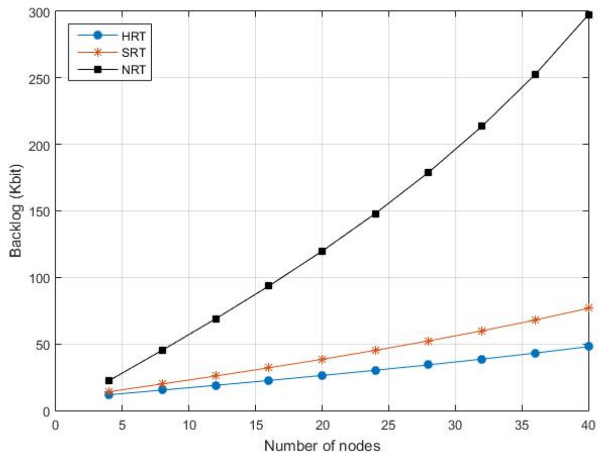


Fig. 8. Upper bounds on backlog vs number of nodes

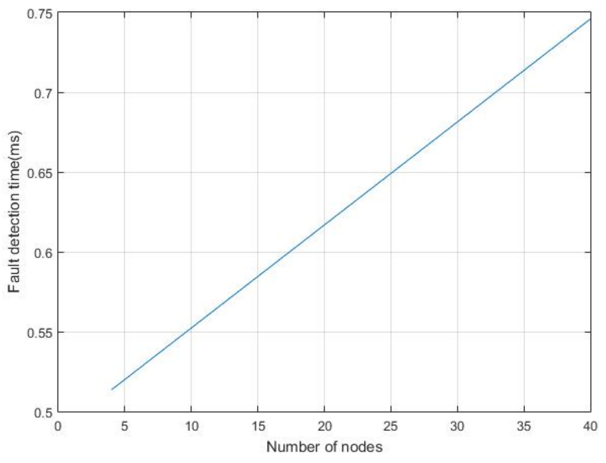


Fig. 9. Maximum fault detection time vs number of nodes

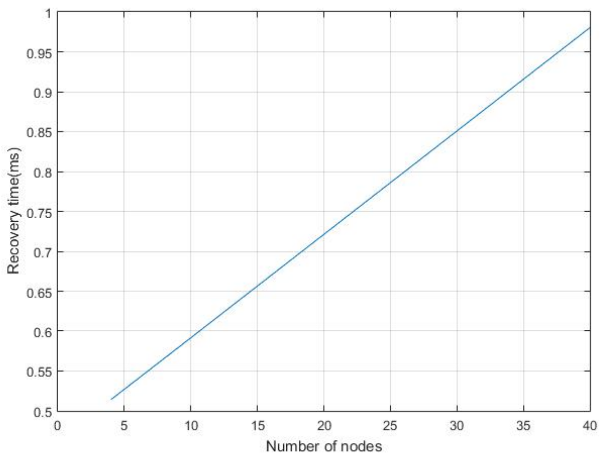


Fig. 10. Maximum recovery time vs number of nodes

These results show the high performance and reliability levels guaranteed by AeroRing for a medium size network.

VI. Conclusion and Future Work

A new RTE solution, called AeroRing, has been proposed in this paper to handle the emerging requirements of new generation aircraft in terms of decreasing the wire complexity and integration costs, and enhancing the real-time performance and availability.

This solution has many advantages, compared to the most relevant RTE solutions supporting ring topology, such as:

- enhancing the **performance** and the resource usage efficiency due to its distributed access mechanism and its QoS-aware routing algorithm;
- offering high **availability** and **reliability** levels through a distributed fault management mechanisms, which avoid any single point of failure;
- improving the network **flexibility** through event-triggered communication support without any need of synchronization;
- minimizing the implementation **costs** due to its compatibility with IEEE 802.3 standard, and the configuration effort through its auto-configuration mechanisms.

AeroRing has been specified to fulfill the avionics requirements, but it can be easily extended for other industrial application fields, such as automation and control. This adaptation will be investigated as a next step of our work. Furthermore, AeroRing consortium is working on the standardization process of such a proposal with open source specifications, to facilitate its adoption in the market.

References

[1] "Avionics Full-Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664, Part 7, Aeronautical Radio."
[2] C. Furse and R. Haupt, "Down to the wire," *Spectrum, IEEE*, vol. 38, no. 2, pp. 34–39, 2001.
[3] *IEC 61784-2, Digital data communications for measurement and control - Part 2: Additional profiles for ISO/IEC 8802-3 based communication networks in real-time applications*, 2010.
[4] M. Felser, "Real-Time Ethernet - Industry Prospective," *Proceedings of the IEEE*, vol. 93, 2005.
[5] E. Wandeler, A. Maxiaguine, and L. Thiele, "On the Use of Greedy Shapers in Real-Time Embedded Systems," *Embedded Computing Systems*, vol. 11, no. 1, 2012.
[6] *IEC 62439-3, Industrial communication networks - High availability automation networks - Part 6: Distributed Redundancy Protocol (DRP).*
[7] *IEC 62439-7, Industrial communication networks - High availability automation networks - Part 7: Ring-based Redundancy Protocol (RRP).*
[8] *IEC 62439-1, Industrial communication networks - High availability automation networks - Part 1: General concepts and calculation methods.*

Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet

Marc Boyer, ONERA, France
Hugo Daigmorte, ONERA, France
Nicolas Navet, University of Luxembourg
Jörn Migge, RealTime-at-Work, France

Abstract: Switched Ethernet is becoming a de-facto standard in industrial and embedded networks. Many of today's applications benefit from Ethernet's high bandwidth, large frame size, multicast and routing capabilities through IP, and the availability of the standard TCP/IP protocols. There are however many variants of Switched Ethernet networks, just considering the MAC level mechanisms on the stations and communication switches. An important technology in that landscape is TTEthernet, standardized as SAE6802, which allows the transmission of both purely time-triggered (TT) traffic and sporadic (or rate-constrained - RC) traffic. To the best of our knowledge, the interactions between both classes of traffic have not been studied so far in realistic configurations. This work aims to shed some light on the kind of performances, in terms of latencies, jitters and useful bandwidth that can be expected from a mixed TT and RC configuration. The following issues will be answered in a quantified manner by sensitivity analysis: How do both classes of traffic interfere with each other? What are the typical worst-case latencies and useful bandwidth that can be expected for a RC stream for various TT traffic loads? What is the overall impact of TTEthernet integration policy for the RC traffic? This study builds on a worst-case traversal time analysis developed by the authors for SAE6802, and explores these questions by experiments performed configurations of various sizes.

Keywords: Time-Triggered Ethernet, worst-case traversal times, sensitivity analysis, benchmarking.

1 TTEthernet: transmission schedule for several classes of traffic

The SAE standard AS6802, [AS6802] describes a network called *Time-Triggered Ethernet*, also known as *TTEthernet* [Ste09]. As explained in the standard, AS6802 "*adds synchronization and time-deterministic data transfer characteristics to those Ethernet operations that use active star (e.g., hub or switch) topologies, while retaining full compatibility with the requirements of IEEE 802.3*" [AS6802]. In fact, while keeping the same frame format as IEEE 802.3, AS6802 defines three kinds of data flows:

- A time-triggered (TT) traffic, where a global (periodic) time schedule defines for each TT flow the time point at which frames have to be sent.
- A rate constrained (RC) traffic, where each flow has a bandwidth limit defined by two parameters: a minimal duration between two successive frames at the source and a maximal frame size. This constraint is the same as the one of ARINC664 P7, also known as AFDX (Avionics Full DupleX ethernet), where this minimal duration is called a BAG (Bandwidth Allocation Gap).
- A best-effort (BE) traffic that is simply a class for low-priority Ethernet traffic without timing and delivery guarantees.

Both TT and RC flows are statically defined, with a single source, a static routing, and a set of receivers.

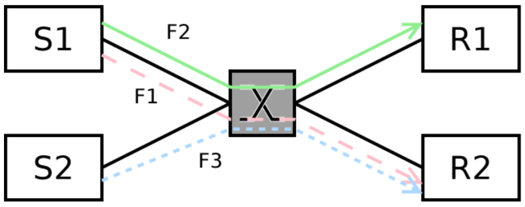


Figure 1: Topology and data flow example.

Let us consider the system shown in Figure 1 with a single switch connecting four nodes, S1, S2, R1, and R2. The node S1 sends two flows, F1 to R2 and F2 to R1, while the node S2 sends one flow, F3 to R2. Let us assume that F2 and F3 have the same period P, and F1 a period equal to 2·P.

Sending all flows as TT traffic requires building a global frame schedule for all links such that there is no contention, and, in the best case, no buffering of frames in switches (a frame is sent as soon as it is received). Such a schedule is given in Figure 2.

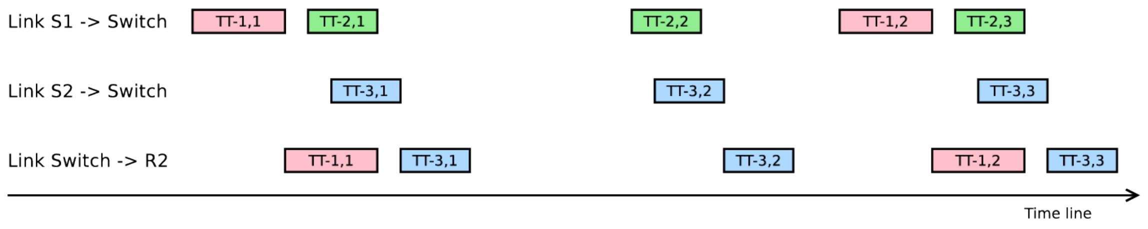


Figure 2: Schedule example with purely TT traffic.

With purely RC traffic, no synchronization is needed between the nodes, what is required is only the respect of the per-flow inter-frame gap in emission. Then, some contention can occur in the switches, and frames have to be buffered. This buffering adds some jitters to the flow. For instance in the example of Figure 3, even if the flow F3 is sent with a periodic pattern, the interference can shorten the distance between two successive messages (see messages RC-3,1 and RC-3,2).

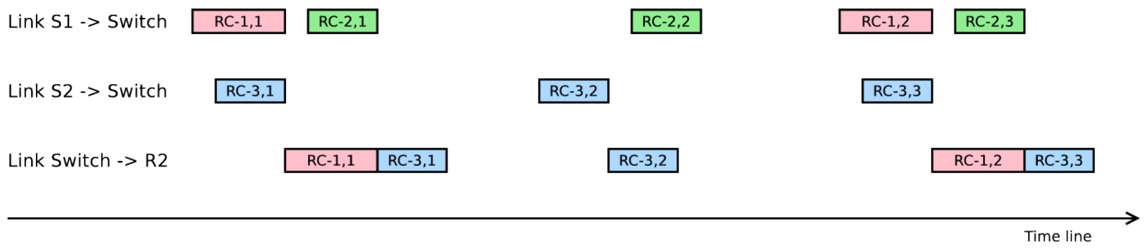


Figure 3: Schedule example with purely RC traffic.

At first sight, TT traffic seems a better solution than RC traffic, but it requires a lot of care. The first obvious point is that it requires a global clock, and a large part of the SAE standard consists in defining a robust protocol for global synchronization. It implies the addition of "protocol control frames" (PCF). A second requirement is the ability to build a global communication schedule, which is a NP-complete problem. Generating communication schedule involves non-trivial optimization algorithms, see for instance [Ta13], whose actual performances are difficult to assess given that no optimal solution can be known except on very small problems. Lastly, the best temporal performances are achieved when synchronizing the tasks schedule and the communication schedule. Actually, the network is meant to transfer data between tasks, and the delay to control is the delay between data production and data consumption. Indeed, without synchronization between the tasks and the network, a data may have to wait a full transmission period before being sent. Similar asynchronisms can create latencies in reception too. This means that, if the local scheduling on a computer is changed during the design process of a system, it may impact the network scheduling and then the scheduling on all computers.

On the opposite, a RC traffic does not require a global synchronization. It also requires some analysis method, not to build the global schedule, but to verify that the system's timing behavior will respect the memory and frame latency constraints (see [Gr04, Fr06, Bo11] for switched Ethernet networks). An important property of the RC traffic is that the frame communication delays can be computed independently of the task scheduling on the sending station. This means that changing the task scheduling on a station will not have any impact on the other nodes as long as long as the constraint of the minimum time between two successive transmissions is met.

Because not all streams have the same transmission requirements, and because development constraints may prevent the use of TT traffic for some nodes, it can be a practical solution to take advantage of the AS6802 protocol flexibility and mix on the same network TT and RC traffic as illustrated on Figure 4. But mixing both kinds of traffic[1] implies interferences. Each TT frame is scheduled for transmission on a link at a specific time point, whereas an RC frame can be sent almost at any time, creating thus interferences at the

[1] The standard actually supports three kinds of traffic: TT and RC, but also Best Effort (BE) traffic. This latter class that comes without any guarantee with respect to delays and even proper frame delivery will be ignored in this paper.

link access level. The SAE standard defines two integration policies for the RC and TT traffic in the communication switches: *shuffling* and *preemption* [AS6802], whereas former works have considered also *timely block* and *resume preemption* [Ste09]:

- In case of *shuffling*, an RC frame can be sent at any time, and the transmission of TT frame is postponed so that the RC frame can complete its transmission. Then, a TT is no more associated to a time instant (called *scheduled point in time*) but to a time window, the *scheduled window*[2].
- In case of *preemption*, when a TT frame has to be sent, the sender aborts the transmission of the RC frame in order to send the TT frame immediately. The RC frame is sent again after the TT frame, from the start (this is called *preemption restart*).
- The *timely block* idea is to block any RC frame emission if its emission time can interfere with the next TT frame.
- The *resume preemption* mechanism consists in resending the frame from where it was stopped, and not from the start. These two latter mechanisms have not been kept in the standard.

In case of preemption, the low jitter and low latency of the TT traffic is favored over the RC one, but some bandwidth is lost and RC traffic latencies increase. On the opposite, in case of shuffling, no bandwidth is lost but the TT traffic will experience a larger latency.

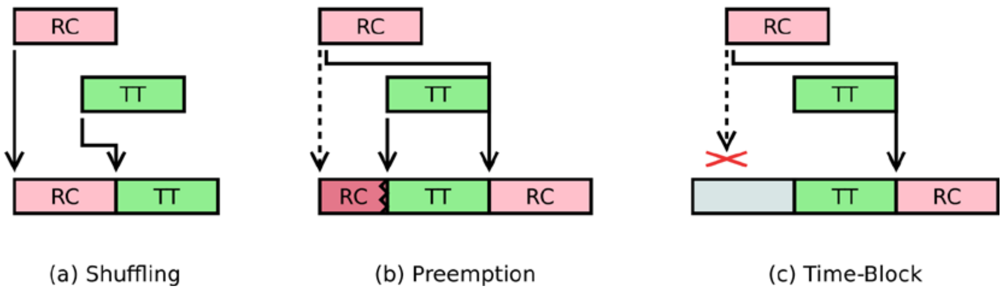


Figure 4: TT-RC integration policies.

Two RC-TT traffic integration policies are considered in this study: timely-block, which minimizes the jitters for the TT flows, and the shuffling integration policy which is work-conserving.

The global clock synchronization service is implemented through the exchange of dedicated frames. The SAE standard uses PCF (Protocol Control Frames) of small size (64 bytes) for this purpose. These PCF frames belong to some specific RC flows, with priority higher than all other flows. To solve contentions with TT frames, the shuffling policy is used. To solve contentions with lower priority RC flows, non-preemptive priority is used. Due to the existence of PCF, a TT frame is assigned a slot whose length is a *time window*, *i.e.* whose length is large enough to contain the TT frame and a PCF frame, and also an RC frame in case of shuffling.

2 Experimental setup

2.1 Worst-case traversal time evaluation

Since the network is one link in the timing chain of a real-time distributed function, its real-time capability must be proven, that is to say, for each frame, an upper bound on the network latency must be guaranteed. The network latency is also often referred to as Worst-Case Traversal Time (WCTT) in the literature.

In the case of a Time Triggered flow, upper bounding the WCTT is quite simple: once the global schedule has been derived, the upper bound on the delay is the distance between the emission time on the first link to the reception time on the last link (cf. Figure 2). The TT transmission schedules in this study have been generated using the TTE-Plan tool from TTTech (TTE-Plan 4.2 for all experiments except the ones with 500VLs which required TTE-Plan 4.3 with specific raster tick settings). For RC flows, there is no global schedule but the BAG contracts enable to upper-bound the workload submitted to the network per flow. This information is used by schedulability analyses that derive an upper bound on the network latency for each flow. The reader is referred to [Fr06, Ba09, He12, Bo12, Gu13, Bo14] for good starting points about the techniques and their performances.

[2] To a lesser extent, this is also the case without shuffling, since one have to deal with limited global clock accuracy.

The WCTT analysis used in this study relies on the network calculus theory [Fr06], which was used to certify the A380 AFDX backbone and is still used in certification today. The pessimism of state-of-the-art implementation has been experimentally evidenced (see [Bo12]) and NC can be extended to account for fine-grained system characteristics such as task scheduling [BD12], frame scheduling at the end-system level [Bo14] and transmission offsets [Li11]. The WCTT analysis used in this work extends [BD12, Li11] by considering the TT traffic as produced by a local scheduler, and adding the impact of the integration policy. Another basic idea underlying the WCTT analysis is to consider the processing and transmission times of TT frames as time periods during the resource is unavailable for RC traffic and adapt the network-calculus service curves accordingly. The analysis is implemented in the RTaW-Pegase timing analysis tool for embedded communication architectures developed by RTaW in partnership with ONERA.

Bounding the WCTT of the RC traffic in presence of TT traffic is also studied in [St11] and [DP15]. In [St11], the author assumes that there is, in each buffer, at most one frame of each crossing VL and computes the delays based on the knowledge of the TT schedule. A more precise analysis is presented in [DP15], based on the notions of busy window, ET availability and ET demand, that are somewhat similar to the concepts of service and arrival curves in network calculus. The maximal number of RC frames in each buffer during a busy window is computed with regard to the VL BAG, neglecting the jitter introduced in the network that must be added to have correct bounds. The contention between two frames sharing some buffers in their path occurs only once in FIFO policy. This effect is called “grouping”, “serialization” or “shaping” in [Ba09] [Fr06] [Bo11] and is also modeled in [DP15] by subtracting the delay introduced by flows sharing two consecutive buffers. The delay introduced by the TT flows is accounted for by enumerating, as possible start of busy windows, all starts of TT frame emission in the global TT schedule.

2.2 Network topologies

Three network configurations of various sizes are considered in the experiments of this paper:

- 4S-200VL: 4 communication switches and 200 rate-constrained multicast flows, called VLs in the following (VL stands for Virtual Links in AFDX terminology),
- 8S-200VL: 8 switches and 200 VLs,
- 8S-500VL: 8 switches and 500 VLs.
- The topology of a configuration is made up of 4 or 8 switches, connected as a 2x2 or 2x4 mesh structure as illustrated on Figure 5. The links data rate is set to 100Mb/s and the switching delay to 1.5µs.

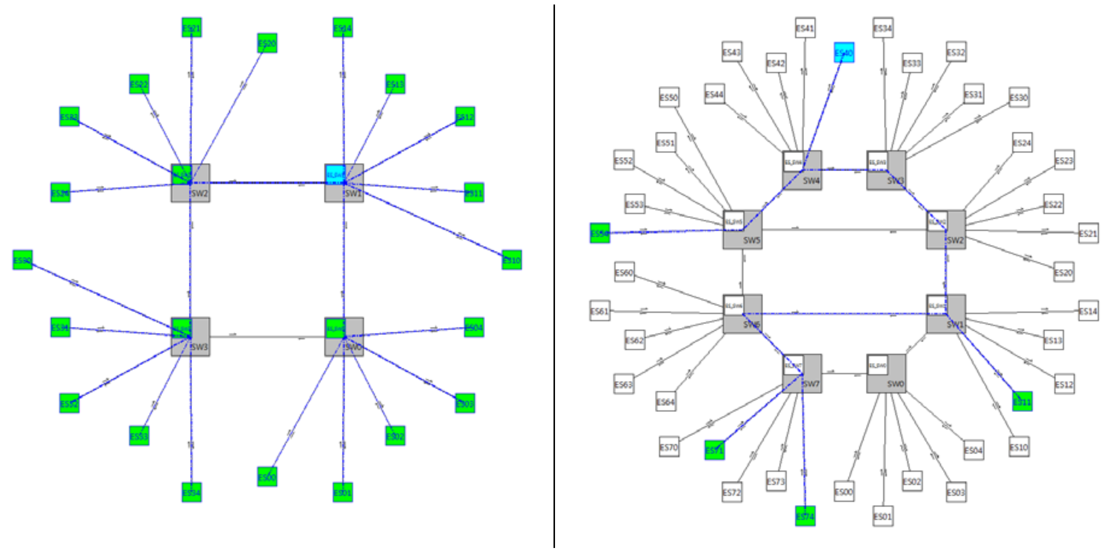


Figure 5: Topology of the case studies. The right-hand figure shows the 4 switches 2x2 mesh topology configuration with a synchronization frame broadcasted by switch 1. The left-hand figure shows the 8-switches 2x4 network configuration with a multicast stream (RTaW-Pegase screenshots).

Five end-systems are connected to each switch. Then, the predefined number of VLs is created with the following procedure for each VL:

- The source node is randomly decided amongst all nodes.

- The number of receivers is randomly chosen between 1 and 5 with a uniform distribution, and the destination nodes are randomly selected (avoiding the source). The expected total number of flows is hence 3 times larger than the number of VLs.
- The size of the frames of the VL is randomly chosen according to a uniform distribution: as many more small frames, but still some large ones. The range of the size distribution is adjusted wrt the network load objective and the number of VLs.
- The frame rate, or BAG, is randomly chosen with a uniform distribution in the set of values authorized by the AFDX standards. The BAG value is biased towards large values. Indeed, with an equiprobable choice, the load generated by larger BAG value flows (e.g., 128ms BAG) would have been negligible compared to the load of the smaller BAG flows (e.g., 2ms BAG).

2.3 Distributions of BAG and frame size

The distributions resulting from the random generation for the BAG and frame size of the VLs is shown in Figure 6 for the 4S-200VL configuration, where the area of the circles is proportional to the number of VLs with these parameter values. For example, the top-left circle shows that there is only one VL with BAG 128ms and maximal size 107bytes, and the large circle near the 16 label shows that they are 29 VLs with BAG 16ms and size 64 bytes.

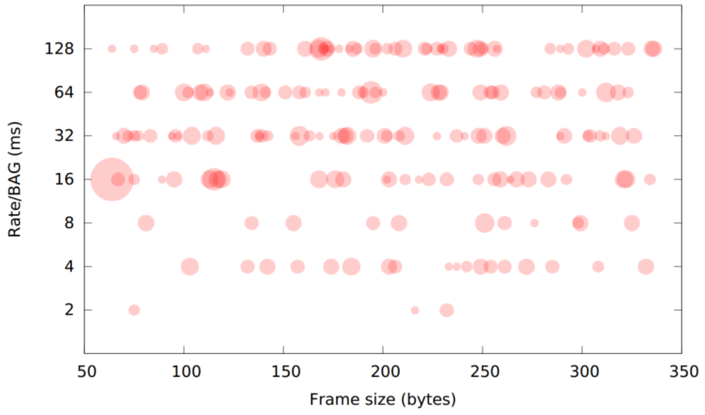


Figure 6: Distribution of BAG and frame size for the 4S-200VL configuration.

The load on the links between the switches in the 4S-200VL configuration ranges from 2.05% to 7.56%, the variability being due to the randomness of the configurations. The same distributions are shown for the 8-switches 200 VLs configuration (8S-200VL) and the 8-switches 500VLs configurations in Figure 7. For these two latter configurations, the load of the links between the switches respectively ranges from 3.31% to 16.68% and from 5.20% to 27.45%. The larger load in these configurations is due to the larger number of VLs but also to the larger maximum frame size (twice the maximum size used to generate the first configuration).

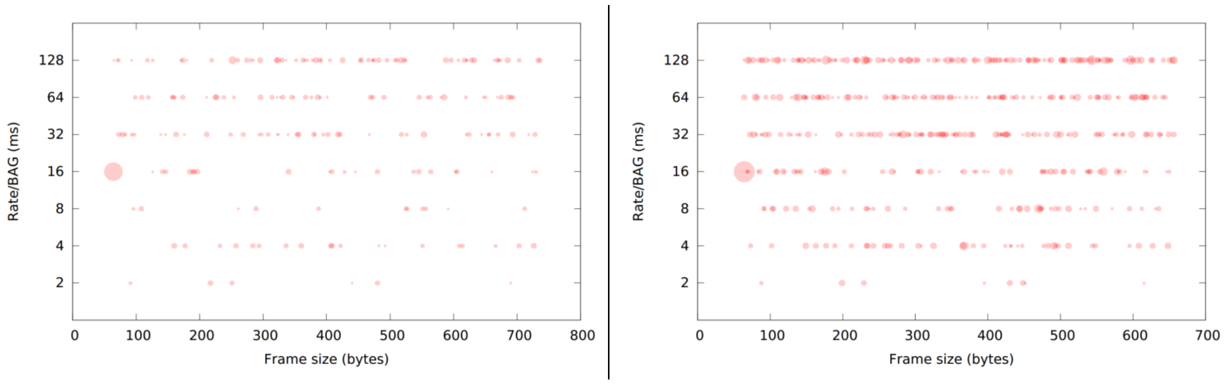


Figure 7: Distribution of BAG and frame size for the 8S-200VL and 8S-500VL configurations.

2.4 Experiments: increasing TT traffic with/without schedule regeneration

The aim of this study is to evaluate the impact of the TT traffic over the RC traffic. This will be assessed by gradually increasing the share of the TT traffic while simultaneously reducing the RC traffic by the same amount. This comes to consider networks where an increasing share of the RC traffic is turned into TT traffic. At each stage, the WCTTs of the RC flows are recomputed.

We first consider a configuration entirely consisting of RC traffic, called “0 TT”. We then split this configuration into 10 subsets of VLs (S1, S2, S3,…, S10) , each subset being made up of 10% of the VLs:

- The VLs in S1 are transformed into TT traffic, the rest of the traffic remaining RC. This configuration is denoted by “S1 TT”.
- On the basis of “S1 TT”, S2 is turned into TT traffic. This configuration is denoted by “S1-2 TT”.
- This goes on until one ends up with a complete TT configuration (“S1-10 TT”), leading overall to 11 configurations.

The frames of all TT flows are assumed to have a deadline constraint equal to the period of the flow. For each traffic configuration under study, two different experiments are performed:

- **Experiments A:** a global schedule is built for the TT flows, the routing is set for all flows of the 11 configurations, and an upper bound on the WCTTs of the RC flows is computed. From the scheduling point of view, each configuration is a new problem submitted to the off-line scheduler TTE-Plan, and there is no link between two flow configurations. In particular, the scheduler will often choose different routes for the same VL in the RC class, as it will be seen later in the experiments.
- **Experiment B:** we start with “S1-S10 TT” where all flows are in the TT class. A global TT schedule is built for this configuration. Then, 10%, 20% and up to 100% of the flows are progressively becoming RC, but the same schedule is kept for the remaining TT flows, and the routing remains the same as in is “S1-S10 TT” for all VLs.

Keeping the same routing and same TT schedule as done in experiment B eases the comparisons but this is not optimal in terms of scheduling performances. Indeed, our observation has been that the off-line TT scheduler tries to spread the TT slots as uniformly as possible over time, creating hence time windows for the RC frames to be transmitted with little delays, and subtracting TT slots at random will not lead to an optimally balanced TT load.

3 Performances of mixed TT and RC configurations – an empirical evaluation

The performance evaluation study is conducted by progressively turning RC VLs into TT VLs and studying the impact on the RC flow latencies. The outcomes of the experiments are difficult to predict because there are different and conflicting effects of the TT traffic over the RC traffic:

1. **Priority change**: in AS6802, the TT flows have a higher priority than the RC flow. Hence, moving a flow from RC to TT will reduce the remaining bandwidth left to the RC flows, and increase their delays.
2. **Loss of bandwidth**: When the integration method is *timely block*, some bandwidth just before a TT window may have to be left unused, increasing thus the delays of the RC traffic.
3. **Contention reduction:** The TT frames are scheduled by the off-line scheduler TTE-Plan which will shape the TT traffic over time. Indeed, the scheduler tries to spread the TT time-windows along the system hyper-period. This is beneficial for RC flows since it reduces the number of frames and thus the waiting times in the communication switch queues.

To illustrate contention reduction, let us consider the topology in Figure 1 assuming that all flows are converging to station R2. The flows F1 and F2 are TT while the flow F3 is RC, and all flows have the same BAG. By shaping the TT load along the hyper-period, the scheduler will insert idle times between the time windows of both TT flows. Hence, the RC frame from flow F3 can be delayed either by a frame of F1 or a frame of F2 but not by both, as illustrated in Figure 8. On the contrary, if the flows F1 and F2 were RC, they could arrive at the switch back-to-back and the RC flow would be delayed by both.

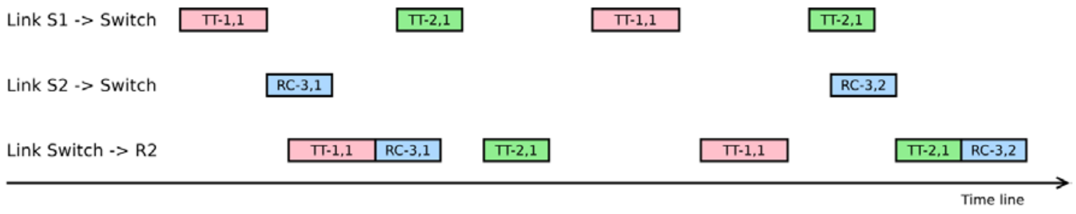


Figure 8: TT shaping reduces contention by idle times by TT time windows.

3.1 Configuration with 4 switches and 200 VLs (4S-200VL)

The graphs on Figure 9 show the worst-case traversal times (WCTT) of the RC flows with shuffling and timely block in Experiments A (communication schedule regeneration at each step). It must be noted that The outliers are due to flows whose routing was changed with respect to their routing in their 0TT configuration. Indeed, the changes of routing negatively impact many RC flows. We observe anyway that globally an increasing share of TT traffic helps to reduce the RC WCTTs (see also Table 1). This holds for both shuffling and timely block. Here, the positive shaping effect outweighs the detrimental effects listed previously as soon as the TT load is above 20%.

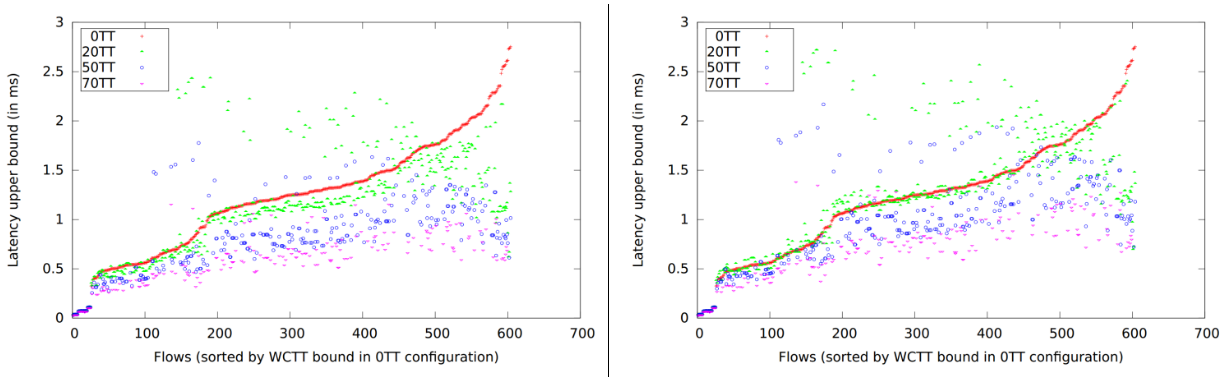


Figure 9 : Upper bounds on the worst-case traversal times (WCTT in ms) for the rate-constrained flows in *Experiments A* on the 4S-200VL configuration with shuffling (left) and timely block (right). The curves show the WCTTs for a share of TT traffic equal to 0% (0TT), 20% (20TT), 50% (50TT) and 70% (70TT). The VLs are sorted by increasing WCTTs in the 0TT configuration.

As can be seen in Figure 10 that shows Experiments B (*i.e.*, no TT schedule regeneration at each step), shuffling is logically better (12.5% on average over all flows) for RC flows than the timely block scheme. In experiments B with shuffling (see Table 1), the average WCTT for RC flows is 1.20ms with 0% TT load, 1.08ms with 20% TT load, 0.82ms with 50%TT load and 0.33ms with 90% TT load. However, this large latency improvement, up to 72% over purely RC traffic, is only for the reduced set of RC flows left.

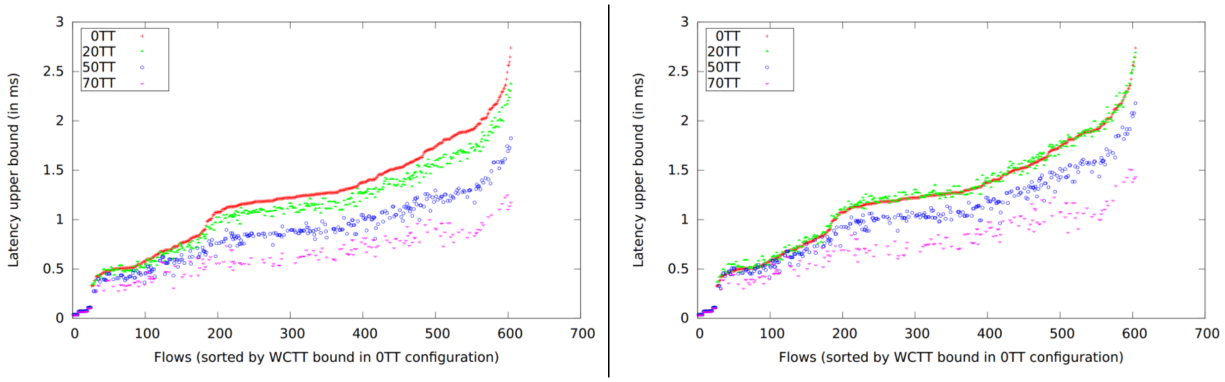


Figure 10 : Upper bounds on the worst-case traversal times (WCTT in ms) for the rate-constrained flows in *Experiments B* on the 4S-200VL configuration with shuffling (left) and timely block (right) for an increasing share of TT traffic (0 to 70%).

Table 1 summarizes the results over all traffic configurations for Experiments A and B. The small difference there is at 0% TT load can be explained by differences in the routing provided by TTE-Plan. On this small system, except in one case, the WCTTs of RC flows decrease monotonously with the increase of the TT traffic. The number in bold in Table 1 shows a case where the loss of bandwidth due to timely block is not fully compensated by the better shaping of the TT traffic.

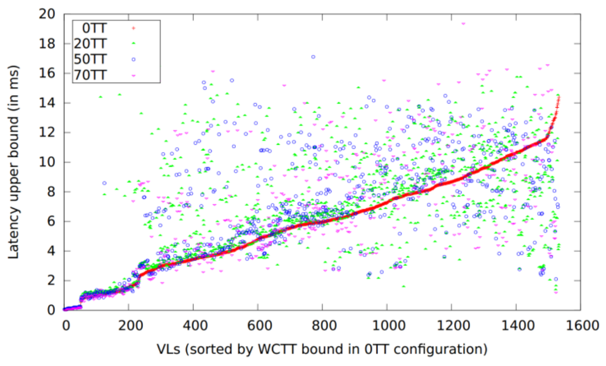
TT traffic	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
A-Shuffling	1.27	1.22	1.17	1.06	0.97	0.86	0.80	0.64	0.52	0.43
A-Timely	1.27	1.30	1.29	1.20	1.16	1.02	0.95	0.76	0.61	0.51
B-Shuffling	1.20	1.16	1.08	0.99	0.90	0.82	0.68	0.59	0.47	0.33
B-Timely	1.20	**1.25**	1.21	1.13	1.05	0.97	0.82	0.70	0.54	0.37

Table 1: Average WCTT in ms of RC flows with an increasing share of TT traffic on the 4S-200VL configuration. 'A-Shuffling' stands for experiments A with shuffling traffic integration mechanism.

Due to space constraints, the results are not shown in this paper for the configuration with 8 switches and 200VLs. The reader is referred to [Bo15] for the complete set of experimental results.

3.2 Configuration with 8 switches and 500 VLs (8S-500VL)

In this section the same experiments are conducted on a larger configuration in terms of topology (number of switches and stations). The frame size is also twice the size used for the smaller system. As a result of that, the network is more highly loaded with links loads up to 27.5% (see §2.3). Figure 11 shows the WCTTs for the RC traffic with timely block on experiments A (left graphic) and B (right graphic). WCTTs obtained with Experiments A form a cloud of points in the left-hand graphic of Figure 7 because, in this more constrained problem, the off-line scheduler defines for the majority of RC flows different routes at the different TT load levels. The gain obtained with TT traffic can best be seen in Figure 12 (left graphic). However, as seen on Figure 11 (right graphic) and Figure 12 (right graphic), on the contrary to the results obtained with the small system, here more TT load leads to degraded performances for the RC traffic when timely block is used. Indeed this mechanism involves a loss of bandwidth for RC frames that increases with the number of TT frames exchanged in the system.



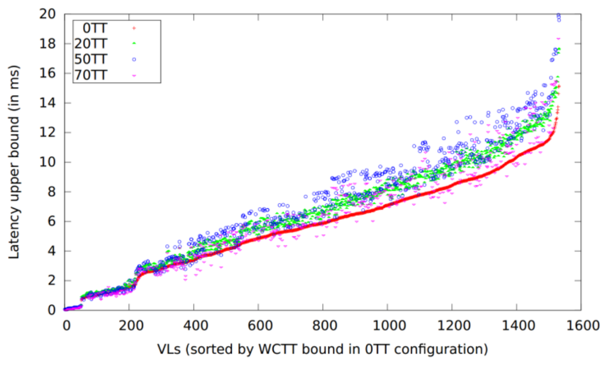
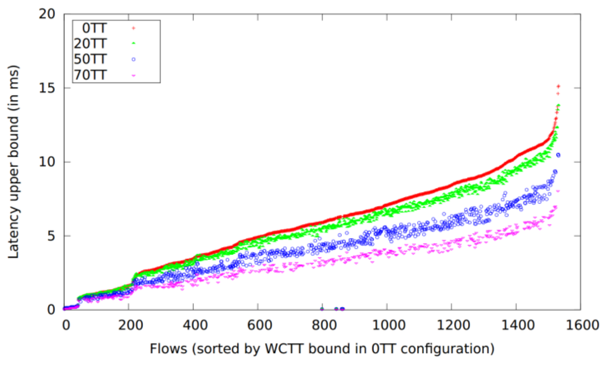


Figure 11 : Upper bounds on the worst-case traversal times (WCTT in ms) for the rate-constrained flows in *Experiments A* and *Experiments B* on the 8S-500VL configuration with timely block (right) for an increasing share of TT traffic (0 to 70%).



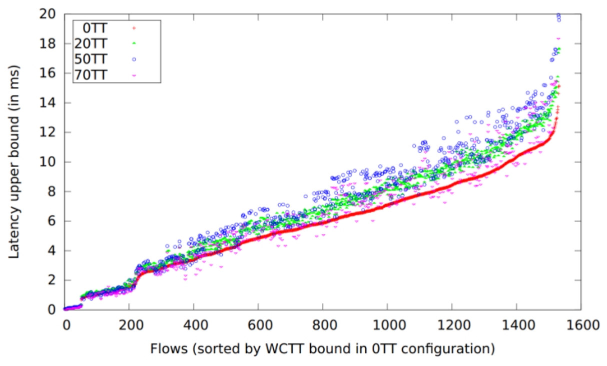


Figure 12 : Upper bounds on the worst-case traversal times (WCTT in ms) for the rate-constrained flows in *Experiments B* on the 8S-500VL configuration with shuffling (left) and timely block (right) for an increasing share of TT traffic (0 to 70%).

What we see in Table 2 is that on this more loaded configuration shuffling clearly outperformed timely block for RC traffic WCTTs. In both Experiments A and B, the WCTTs of RC flows steadily decrease with an increasing share of TT traffic with shuffling, while, with timely block, RC flows have larger latencies with TT traffic up to 80% of TT traffic. The larger the TT load, the higher the performance difference between shuffling and timely block. Indeed, the average WCTT response times of RC frames are more than 2 times larger with timely block above 50% of TT traffic in Experiments A and B.

TT traffic	**0%**	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
A-Shuffling	6.07	5.84	5.35	5.16	4.50	4.21	3.72	3.27	2.63	1.83
A-Timely	6.07	**6.54**	**6.79**	**7.40**	**7.00**	**6.87**	**6.93**	**6.85**	6.06	4.63
B-Shuffling	5.76	5.57	5.26	4.92	4.44	3.95	3.60	3.01	2.33	1.52
B-Timely	5.76	**6.33**	**6.57**	**6.94**	**6.69**	**6.82**	**6.60**	**5.99**	5.35	4.40

Table 2: Average WCTT in ms of RC flows with an increasing share of TT traffic on the 8S-500VL configuration. ‘A-Shuffling’ stands for experiments A with shuffling traffic integration mechanism.

In our experiments on the large configuration with 8 switches and 1500 flows, we observed that, when using timely-block, scheduling a share of the flows as TT traffic could degrade the latencies of the RC flows. This degradation is however limited to 21% in our experiments. This same behavior could not be reproduced with shuffling which however leads to larger worst-case jitters for TT flows.

4 Discussion and future work

This study is to the best of our knowledge the first providing an evaluation of the impact of the TT traffic over the RC traffic in an AS6802 network mixing TT and RC traffic with different traffic integration policies. Experiments have been presented, which offer some insights about the interferences between both classes of traffic:

- The timely block integration has a detrimental impact on the RC traffic latencies with respect to shuffling, with an increase of more than a factor 2 over shuffling above 50% of TT flows.
- TT flows, as scheduled with TTE-Plan, tend to spread the network transmissions over time as a traffic shaping policy would do, which reduces the RC latencies. However this only holds with shuffling which incurs increased jitters for TT flows.

These results should however be interpreted with caution for the following reasons:

– The results obtained depend on the global TT schedule. The experiments presented here have been done using the TTE-Plan scheduler of TTTech. Another scheduler, or a modified version of TTE-Plan, may lead to different results. The performance of a Time-Triggered system depends on the ability to build an efficient global schedule; in others words, the performances of an AS6802 network depends not only on the network technology but also importantly on the configuration tool.

– Other choices for the network configurations used in the experiments (probability distributions for the random generation, assumption on the clock-drifts, topology choices, etc) may possibly have lead to different findings.

– We built with TTE-Plan a schedule considering only the *shuffling* integration method and used it also for timely block configuration. Our analyses on *timely block* are thus done using a schedule designed for the shuffling method.

– The algorithms computing the upper bounds on delays for RC frames only provides upper bounds, not the exact worst case which is unknown. There is currently no method to compute tight lower bounds on the worst-case delays, as it exists for AFDX [Ba09]. Hence there is no way to estimate the actual pessimism of our upper bound in a satisfactory manner. Although, experiences with previous similar analyses in Network Calculus (e.g. [Bo12]) suggest that the analysis should be accurate, this remains to be ascertained.

– At moderate load, the RC traffic benefits from the traffic shaping of the TT traffic, which is facilitated by the assumption that deadlines equal periods. On more constrained systems, with deadlines less than periods, it is possible that this beneficial effect will be less pronounced.

SAE6802, with its three classes of traffic, offers a lot of flexibility in terms of how the communication can be organized. It becomes however difficult for the system designer to know beforehand the impact of configuration choices on the communication latencies. This study is a step towards a better understanding the behavior of SAE6802 and the influence of configuration parameters. Our aim is also to conceive and implement the toolset that will help automate the configuration and verification of SAE6802 networks. Given the number configuration choices parameters involved, we believe that design space exploration techniques that would guide the designer would help to raise the level of abstractions and lead to a faster and more secure design process. Ultimately, this work contributes to a better understanding of how to best integrate mixed-criticality traffic in complex networked embedded systems as currently investigated in the DREAMS FP7 EU project [Dr15].

Acknowledgments: This work has been partially funded by the FP7-ICT integrated project DREAMS (FP7-ICT-2013.3.4-610640).

5 References

[AS6802] "Time-Triggered Ethernet", SAE standard AS6802, 2011.

[Ba09] H. Bauer, J.L. Scharbarg, C. Fraboul, Christian,"Applying and optimizing Trajectory approach for performance evaluation of AFDX avionics network", 14th IEEE ETFA, Palma de Mallorca, 2009.

[Bo11] M. Boyer, J. Migge, M. Fumey, "PEGASE, A Robust and Efficient Tool for Worst Case Network Traversal Time",SAE 2011 AeroTech Congress & Exhibition, Toulouse, France, 2011.

[Bo12] M. Boyer, N. Navet, M. Fumey (Thales Avionics), "Experimental assessment of timing verification techniques for AFDX", Embedded Real-Time Software and Systems (ERTS 2012), Toulouse, France, 2012.

[Bo14] M. Boyer, L. Santinelli, N. Navet, J. Migge, M. Fumey, "Integrating End-system Frame Scheduling for more Accurate AFDX Timing Analysis", Embedded Real-Time Software and Systems (ERTS 2014), Toulouse, France, 2014.

[Bo14] M. Boyer, H. Daigmorte, N. Navet, J. Migge, "Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet", ONERA Technical Report, 2015.

[BD12] M. Boyer, D. Doose, "Combining network calculus and scheduling theory to improve delay bound", 20th Int. Conf. on Real-Time and Network Systems (RTNS2012), 2012.

[DP15] D. Tamas–Selicean, P. Pop, W. Steiner "Timing analysis of rate constrained traffic for the TTEthernet communication protocol", 18th IEEE Int. Symp. on Real-time Computing (ISORC), 2015.

[DR15] Home page of the DREAMS (Distributed REal-time Architecture for Mixed Criticality Systems) FP7 EU project, http:// http://www.uni-siegen.de/dreams/home/, retrieved 13/11/2015.

[Fr06] F. Frances, C. Fraboul, J. Grieu, "Using Network Calculus to optimize AFDX network", Proc. of the 3thd European congress on Embedded Real Time Software (ERTS06), 2006.

[Gr04] J. Grieu, "Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques", Doctoral dissertation, Institut National Polytechnique de Toulouse, 2004.

[Gu13] JJ. Gutierrez, J.C. Palencia, M.G. Harbour, "Response time analysis in AFDX networks with sub-virtual links and prioritized switches" Proc of the XV Jornadas de Tiempo Real, Santander, Spain, 2012.

[He12] E. Heidinger, N. Kammenhuber, A. Klein, G. Carle, "Network Calculus and mixed-integer LP applied to a switched aircraft cabin network", Proc. of the IEEE 20th Int. Workshop on Quality of Service (IWQoS), 2012.

[Li11] X. Li, Xiaoting, J.L. Scharbarg C. Fraboul, F. Ridouard, "Existing offset assignments are near optimal for an industrial AFDX network" Proc. 10th Int. Workshop on Real-tTme Networks (RTN 2011), Porto, Portugal, 2011.

[Na15] N. Navet, J. Seyler, J. Migge, "Timing verification of real-time automotive Ethernet networks: what can we expect from simulation?", Presentation at the SAE World Congress 2015, "Safety-Critical Systems" Session, Detroit, USA, April 21-23, 2015.

[Se15] J. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, "Formal Analysis of the Startup Delay of SOME/IP Service Discovery", DATE 2015, Grenoble, France, March 9-13, 2015.

[Ste09] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "TTEthernet Dataflow Concept", Proc. of Eighth IEEE International Symposium on Network Computing and Applications, 2009.

[St11] W. Steiner, "Synthesis of Static Communication Schedules for Mixed-Criticality Systems", 1st IEEE Workshop on Architectures and Applications for Mixed-Criticality Systems (AMICS), 2011.

[Tam13] D. Tämas-Selicean, P. Pop, W. Steiner, "Synthesis of Communication Schedules for TTEthernet-Based Mixed-Criticality Systems", 10th Int. Conf. on Hardware/Software Codesign and System Synt., 2013.

A Symbiotic Approach to Designing Cross-Layer QoS in Embedded Real-Time Systems

Florian Greff*†, Eric Dujardin*, Arnaud Samama*, Ye-Qiong Song† and Laurent Ciarletta†
*Thales Research & Technology - Palaiseau, France
{florian.greff — eric.dujardin — arnaud.samama}@thalesgroup.com
†LORIA Research Lab - University of Lorraine - Nancy, France
{ye-qiong.song — laurent.ciarletta}@loria.fr

Abstract*—Nowadays there is an increasing need for embedded systems to support intensive computing while maintaining traditional hard real-time and fault-tolerant properties. Extending the principle of multi-core systems, we are exploring the use of distributed processing units interconnected via a high performance mesh network as a way of supporting distributed real-time applications. Fault-tolerance can then be ensured through dynamic allocation of both computing and communication resources. We postulate that enhancing QoS (Quality of Service) for real-time applications entails the development of a cross-layer support of high-level requirements, thus requiring a deep knowledge of the underlying networks. In this paper, we propose a new simulation/emulation/experimentation framework, ERICA, for designing such a feature. ERICA integrates both a network simulator and an actual hardware network to allow implementation and evaluation of different QoS-guaranteeing mechanisms. It also supports real-software-in-the-loop, i.e. running of real applications and middleware over these networks. Each component can evolve separately or together in a symbiotic manner, also making teamwork more flexible. We present in more detail our discrete-event simulation approach and the in-silicon implementation with which we cross-check our solutions in order to bring real performance aspects to our work. We also discuss the challenges of running real-software-in-the-loop in a real-time context, i.e. how to bridge it with a network simulator, and how to deal with time consistency.

***Keywords:* mesh, network, real-time, QoS, simulation, RapidIO**

I. Introduction

We describe the background to our study, and explain our need to build a framework that allows real-software-in-the-loop to be run over both a simulated network and an in-silicon platform.

A. *Embedded Real-Time Systems: a Dynamic Approach*

Embedded real-time systems are part of many application domains. As a result of our background, we have closely observed their developments in transportation domains such as avionics, vetronics and UAVs. We have observed a growing tension between application needs and their execution constraints. We believe that these trends possibly apply to other domains.

On the one hand, we see an evolution of needs in terms of computing power and communication between applications. This is due to the changing qualities of sensors, whose data processing needs are increasing (for example in radar applications), and the emergence of new application categories

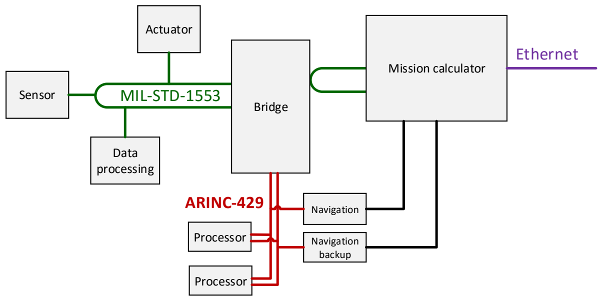


Fig. 1. Static system with real-time buses

such as multi-sensors. Interaction between applications and sensors is increasing, while features tend to be spread over several computing units. Communication architecture becomes more complex. Existing systems mainly process sensor data at high level (e.g. radar tracks). Multi-spectral imagery and cooperation between radar and electro-optical imagery would derive benefit from processing data coming directly from the sensors instead of the tracker. Image processing algorithms could be tuned in real-time in case of unconfirmed radar detection, in order to improve overall performance.

On the other hand, there are stringent constraints relating to hard real-time, criticality (endangerment of the mission or human lives in case of malfunction) as well as material constraints such as size, weight and power (SWaP). When data exchanges between components have hard real-time constraints, network access is typically budgeted (e.g. with TDMA, or static virtual channels in ARINC-664p7) and real-time buses such as MIL-STD-1553B or ARINC-429 are used, as illustrated in Figure 1. Using these buses brings several limitations in terms of throughput and dynamicity (i.e. online dynamic reconfiguration).

We believe that mesh networking of the components of such systems (Figure 2) would reconcile their constraints with the new needs of applications. The plurality of communication paths should result in increased flexibility, resilience, scalability and load balancing characteristics. To improve overall system resilience, it must be possible to redeploy a processing function running on a faulty computing unit to a non-faulty one or operate a graceful Quality of Service (QoS) degradation (Figure 3).

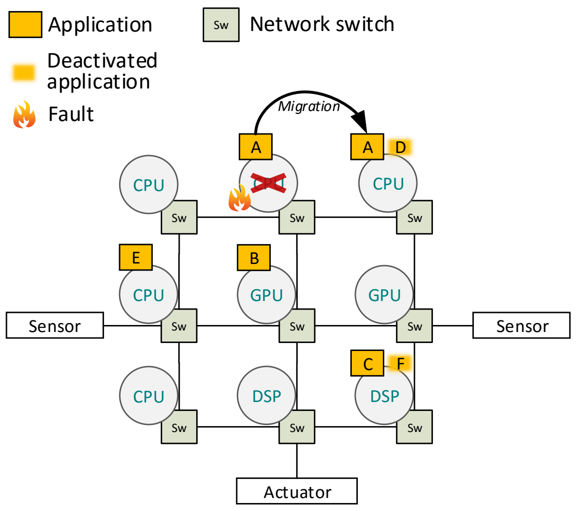


Fig. 2. Dynamic mesh networking of components and applications

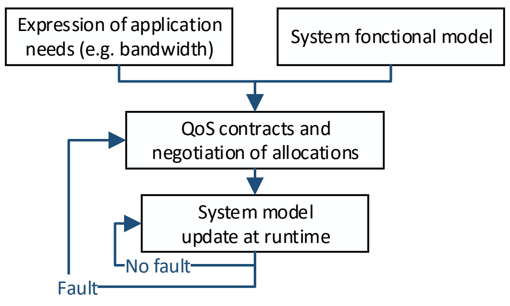

Expression of application needs (e.g. bandwidth)
System fonctional model
QoS contracts and negotiation of allocations
System model update at runtime
No fault
Fault

Fig. 3. Basic working of the dynamic system

For this project, we have chosen, for the time being, RapidIO$^{®}$ [1] [2] as the mesh networking technology. RapidIO was initially designed for embedded systems. It is used in most 3G/4G base stations, medical imagery and high performance computing. Its goals are a very high throughput and guaranteed low latency while providing reliable communication. Low latency is achieved thanks to Control Symbols that can be embedded within packets. They can be seen as small protocol headers used for flow control. The ability to send these Control Symbols even during a packet transmission allows very fast error recovery and thus decreases average end-to-end latency. RapidIO also allows assigning of priorities or virtual channels to flows. However, the way these priorities and virtual channels are used for switching is to be defined by the user. End-to-end flow control can be achieved in multiple ways (credit-based, XON/XOFF, etc.). RapidIO also specifies the definition of register fields for the endpoints and switches. They are used for switch configuration and monitoring. Finally, the standard specifies a clock synchronization protocol, a network discovery algorithm and a very fast fault recovery mechanism. Some additional information is given in Table I.

TABLE I. SOME CHARACTERISTICS OF RAPIDIO

Frame size	1-280 Bytes
Payload size	1-256 Bytes
Maximum bandwidth for four lanes	40 Gbps
Network layer	Cut-through routing support Store-and-forward
Routing protocol	Implementation dependent

B. *Expressing QoS Needs at Network Level*

Achieving this dynamicity in a reliable way requires knowledge of the deadline requirements between the different components. We assume that a good way of dealing with these requirements is a contract-based approach [3] [4]. Contracts allow identification, at runtime, of which network and processing resources are allocated to each application. Our goal is to make this easier by providing cross-layer QoS support from a middleware down to the hardware network switches, thanks to an adaptation layer.

To specify the application QoS (e.g. relative deadline), we consider the use of a Data Distribution Service (DDS) middleware [5]. DDS is a machine-to-machine middleware standard from the Object Management Group, based on the *publish-subscribe* paradigm. It aims to provide a scalable, high-performance and dependable real-time data exchange API. It addresses the needs of various domains such as aeronautics, defense, big data and telecommunications. Publish-subscribe communication seems to fit our needs well, since we aim to support data-intensive applications that can be distributed across multiple nodes (radar for instance). Although DDS provides an interesting way of allowing the QoS requirements specification, there is no underlying mechanism for providing any real-time QoS guarantee, to the best of our knowledge. DDS is indeed designed to run over any type of network and limits itself to indicating the deadline meeting ratio, for instance.

For the implementation of DDS, we have selected Vortex OpenSplice. It was initially developed by Thales Naval Nederland and is now under the auspices of PrismTech as Vortex OpenSplice [6].

To provide full real-time cross-layer QoS support, a complete understanding of network-induced delays is required: buffering, arbitration logic, propagation delay, error recovery based on probabilistic model, contention, etc. We can then design an adaptation layer within the middleware that will translate QoS parameters expressed at application level into what is needed at network level to enforce them (e.g. via packet headers contents).

These assumptions lead us to two major questions:

1) How do we translate QoS defined at contract level into QoS at network level?

2) How can we make use of network parameters (routing, output port packet scheduling, flow control) to reach the desired QoS?

To answer these questions, we first need to build a tool that allows us to implement and test those QoS translation and network-level QoS handling mechanisms. The goal of this paper is to present our symbiotic approach to achieving this goal. This will, in turn, allow us to further answer the above two questions in our future work.

C. The Need for Experimentation

We aim to assess the network properties required to achieve real-time QoS cross-layer support within a mesh network (i.e. with dynamic tasks and network resource re-allocations). For this purpose, we have built a test platform based on the open Serial RapidIO standard, of which an open-source VHDL implementation is available. Having an in-silicon platform is ideal for assessing our final solutions in a real context. However, the complexity of the implementation and the fact that the hardware architecture cannot be changed easily make it difficult to test a lot of algorithms in order to compare them. For that reason, we consider it very useful to work on the key features of cross-layer QoS support (switching, communications scheduling, graceful degradation, etc.) in the context of a network simulator, before assessing them on the real platform.

On the one hand, building the whole system including application code, middleware, OS, CPU/GPU and RapidIO mesh network results in a total experimental approach. On the other hand, simulating all these components would require a great deal of abstraction (at the expense of losing some features) to obtain simulation models of them, whilst it is well-known that modelling applications/middleware/OS/CPU is a difficult task in general.

As a result, we propose to keep using the same upper layers (application code, middleware, OS, CPU/GPU) while having the choice of working on the simulated network or the real one. This allows us to reuse the same set of applications for both networks, thus making implementation and evaluation of cross-layer QoS support and fault-tolerance features easier. It also makes simulation assessment more relevant. Indeed, the only layer to be switched is the network one, so we know that applications and middleware which we would otherwise have simulated do not cause bugs or bad behaviour that we would then have to fix. To the best of our knowledge, there is no experimentation framework able to support such an interaction between real software including middleware, simulated networks and in-silicon ones.

The main contribution of this paper is to propose our symbiotic simulation/emulation/experimentation architecture, ERICA[1], that we use to design real-time QoS support. Applications and DDS middleware can both run above real-world networks and network simulators, thanks to an adaptation layer that we have defined and implemented. We also explain how we have developed a coarse-grain discrete-event simulator achieving the dual goal of accuracy for computing transmission time, and efficiency for running along with operational code. The Ptolemy II modeling framework [7] was used for this purpose. The simulator was tuned according to the run-time platform and then used to help design the latter. Finally, we show the benefit of such a symbiotic approach.

To achieve the interaction between application execution and simulator run, two technical challenges must be solved. First, how to link one application execution to a simulator? Second, how to synchronize the time of the application execution to that of the simulator? This paper is dedicated to answering the first question. We will address the second one in further research; it is sufficient for now to note that this synchronization is not related to real-world time, but to the consistency between the latencies perceived by real software and those emulated by the simulator.

The remainder of the paper is organized as follows. Section II reviews some existing multi-simulation approaches that combine either simulation with hardware-in-the-loop, or two or more different simulators, or one emulator and one simulator. Section III describes the ERICA architecture. Section IV illustrates the results of our approach. Section V discusses the current state of ERICA and presents our ongoing work. We then provide our conclusions in section VI.

II. Related Work on Multi-Simulation Approaches

The idea of using a simulator to emulate part of a more complex system has been highlighted by parallel and distributed simulation (PADS)[4] [8]. PADS addresses the need to distribute a comprehensive simulation, in order to obtain speed-up – by parallelization of computation tasks –, reuse existing simulation components or utilise geographically distributed design environments (e.g. for military training).

High Level Architecture (HLA) [9] is the leading standard for bridging heterogeneous components into an overall experimentation architecture [10] [11]. It is driven by the need to couple existing simulators or *live players* (i.e. users who can interact with the simulation at runtime) without having to design a complete architecture from scratch [12] [8]. It allows hardware to be brought into the loop thanks to live players interfaces. It also specifies a way of achieving time synchronization while federated components may have heterogeneous timing constraints (conservative, optimistic, etc.) [10] [13]. Ptolemy II extensions for its integration within an HLA federation have been designed [14]. However, HLA appears to be very heavy, complex and time consuming when used for designing an overall architecture [11]. Moreover, there is no standard use pattern for it, leading to the need for additional interoperability packages [15] [16].

MECSYCO [17] combines the Discrete EVent System Specification (DEVS) [18] formalism and multi-agent concept to enable the integration of heterogeneous formalisms and manage simulator interoperability. It is notably used for smart grids simulation [19], with existing simulators, although Ptolemy II is not supported yet. However, it does not allow hardware to be brought into the loop and is thus not suitable for our project, since we want to use an in-silicon platform to assess the network solutions at different steps of their design.

COOJA [20] is a wireless sensor network simulator. It includes a microcontroller emulator (e.g. MSPSim for emulating

[1]ERICA stands for ExpeRImenting Cross-layer QoS for Autonomic systems

MSP430 MPU) and a simple radio channel simulator, so that the same sensor source code and OS can run indifferently in COOJA or in a mote (e.g. a MSP430-based TelosB mote). This approach is interesting but slightly different to ours, which runs the application and middleware code over actual hardware machines and a simulated network.

Symbiotic simulation is a more general paradigm in which a simulation and physical systems are closely associated with each other [21]. It is mostly used in the context of "what-if" analysis, i.e. runtime control of the hardware through a simulator that analyzes several scenarios. It also brings the idea that data from physical sensors can improve the accuracy of the simulation. This idea is central to our approach. Indeed, we advocate that data captured from real systems are key both to improving the accuracy of simulation, and to instrumenting the simulation while searching for an appropriate system design.

III. The ERICA Framework: Simulation/Emulation/Experimentation

A. Overall Description

Since we have to work at two different levels (i.e. middleware and network), we need an experimentation architecture and associated tools that allow us to:

- Test our DDS adaptation layer over a well-controlled network
- Design and test network features (e.g. routing or scheduling algorithms) that will be used for QoS enforcement
- Validate our implementation with respect to the expected behaviour of network components such as switches

We have defined a two-layer approach for this purpose, ERICA (Figure 4):

- The high layer is made of real benchmarking applications and middleware (DDS)
- The low layer is the network layer, either real or simulated

The adaptation layer, which will be described below in more detail, allows the same applications and middleware to be run over different kind of networks or simulators at the lower layer.

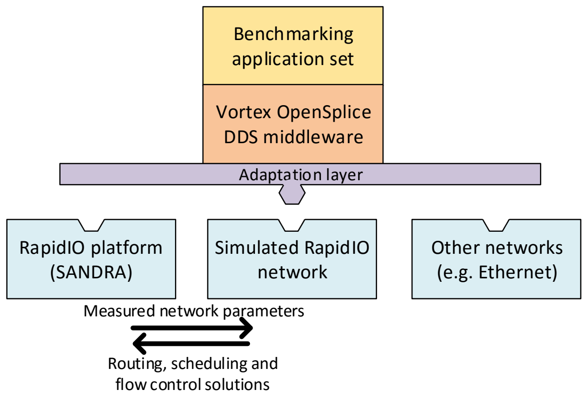

Benchmarking application set
Vortex OpenSplice DDS middleware
Adaptation layer
RapidIO platform (SANDRA)
Simulated RapidIO network
Other networks (e.g. Ethernet)
Measured network parameters
Routing, scheduling and flow control solutions

Fig. 4. ERICA: a two-layer experimentation architecture

The simulated network is used:

- To model a RapidIO network in order to test out routing, scheduling and flow control algorithms (see C)
- To model simple networks in order to test out the way our system translates QoS parameters expressed at the DDS level into network configuration or packet header contents (see D)

The in-silicon platform (see B), used as a reference, gives us some reference values to inject into the simulator (e.g. link latencies), so that we can experiment our features (reconfiguration, routing, etc.) in a representative environment (Figure 4). We then compare its behaviour with that of the simulator, in order to validate both the simulation model and the implementation (see section IV for more details).

B. Experimentation: a RapidIO Platform – SANDRA

Our in-silicon RapidIO experimental platform, *SANDRA*[2], is based on an open-source version available from Open Core [22]. This open-source project, founded by Bombardier® [23] in 2013, provides the physical and logical layers for Serial RapidIO (SRIO), but also one implementation of a SRIO switch.

In order to have a fully observable network adapter, we used this foundation on a Zynq ZC7045 board. The Xilinx® Zynq® programmable SoC [24] [25] [26] is made of an FPGA and a dual-core ARM® [27] Cortex-A9 CPU on the same die. The FPGA and the ARM are tightly connected through several buses, notably 4 high-performance AMBA AXI® [28] slave interfaces and 4 general purpose AXI interfaces (2 slaves, 2 masters). We are using the FPGA to implement our network adapter. It is made of a 5-port SRIO switch. One of the ports is connected to a SRIO-to-AXI bridge, and then connected to one of the High Performance AXI ports of the ARM CPU. We are running a Linux partition on top of a hypervisor. This partition contains our test application running on top of DDS. An illustration of the overall platform is given in Figure 5.

[2]SANDRA stands for Self-Adaptive Network for Distributed and Reconfigurable Applications

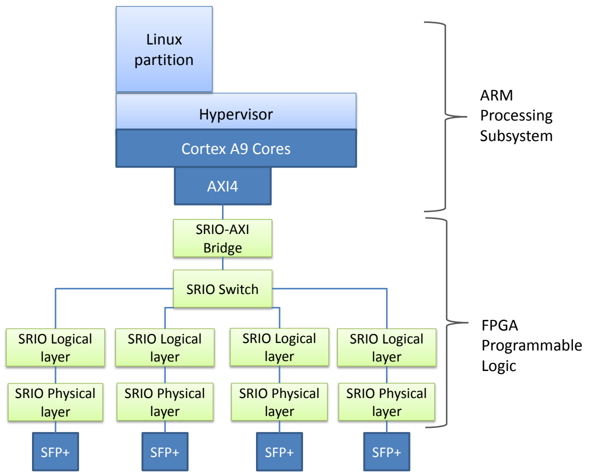


Fig. 5. SANDRA architecture

If we consider a typical network adapter or switch, the internal logic and arbitration is not known. In our case, thanks to the VHDL source availability, we are able to understand and investigate any behaviour.

The FPGA design tools provide precise timing measurements. First, they can run cycle-accurate simulations of the design. We can feed them with test patterns such as single packets, single flows, then simultaneous packet arrivals, link errors, arbitration logic changes, etc. The resulting time measurements are key to testing our design, to pinpointing the root cause of possible erroneous behaviour observed at macroscopic level (e.g. unexpected latencies), and reliably estimating the latency associated with each logical block. These design tools also allow insertion of probes to record true in-silicon measurements at runtime, so as to confirm the results of this cycle-accurate simulation.

C. Network Simulation: an Example Using Ptolemy II

1) Architecture of the Simulation Model: We are designing this simulator using a discrete-event model. This has been shown as appropriate for network simulation [29] [30]. We are simulating the functions of SRIO adapters, and our references are the SRIO standard [1] and the VHDL implementation, especially the use of the same buffering along the transmission path. This work has been done on the Ptolemy II modelling framework, developed at the University of California Berkeley [7].

The goal was to keep our design modular enough to avoid complexity. While basic elements like FIFOs and buses – that we are not willing to work on once the model is validated – are modelled in a strong though more complex way (i.e. graphic), Python scripts are used for simulating routing and scheduling elements. Such a programming language brings more expressivity and flexibility for the key features related to cross-layer QoS support, which we need to change and test frequently, while the rest of the model remains as visually clear as possible.

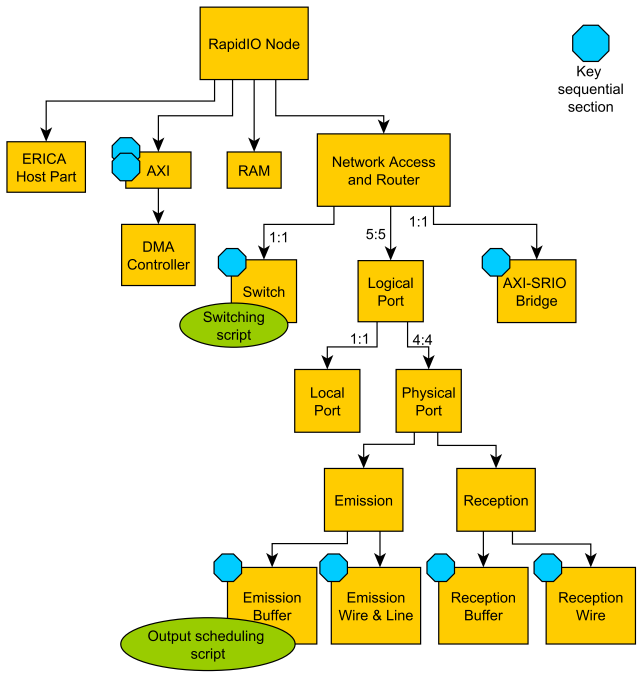


Fig. 6. Composition hierarchy of the RapidIO model

In order to avoid pointless complexity, our RapidIO model focuses on the fonctions which have a significant impact on determinism, flow-control and routing. Indeed, we are not aiming at very precise measurements, but only at transmission times accurate enough to let us work on cross-layer QoS support in a representative environment. As we will detail in section IV, measurements from the platform allow us to assess this accuracy.

We identified *key sequential sections* within RapidIO nodes, in which the individual latencies of logical blocks can be gathered into a *global latency*. This applies when the transition across them is deterministic, i.e. there is no protocol to model between the beginning and the end of the section. Obviously we retain only the sections whose global latency is significant enough to be considered. These are:

- RAM to switch via AXI Bus and Bridge, for read actions
- Switching delay
- Emission/reception buffers to switch and vice versa
- Emission/reception buffers to line and vice versa
- Switch to AXI-SRIO Bridge, for write actions
- AXI-SRIO Bridge to RAM via AXI Bus, for write actions

The composition hierarchy of our model is shown in Figure 6. Note that it is a logical view of the platform, not a direct representation of it.

Once the simulator behaviour has been validated with respect to the SRIO standard, latencies are tuned thanks to the in-silicon platform and its cycle-accurate simulator. For

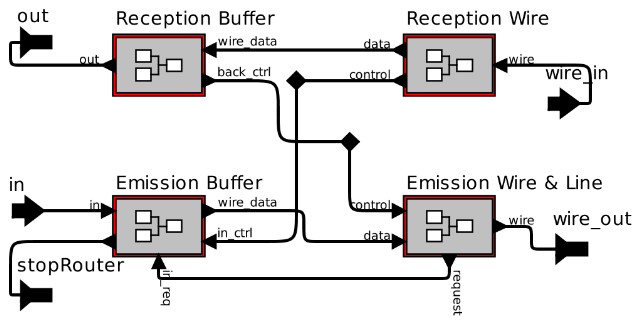


Fig. 7. Structure of the Port actor

each sequential section, we measure the traversal time for a set of packet sizes. We then identify a formula that computes the traversal time of the sequential section depending on the packet size, close enough to the measurements. More details about this process are given in section IV.

To illustrate our approach, we focus on the `Physical Port` actor. This actor is composed of four components, since it distinguishes between, on the one hand, the inbound and outbound traffics, and, on the other hand, the wire level and the packet buffering level, as depicted in Figure 7:

- At wire level, the `Emission Wire & Line` and `Receive Wire` actors handle sending of packets and Control Symbols. In RapidIO, Control Symbols may interrupt packets. For this reason, we send packets as two events (packet beginning, packet end), while Control Symbols can be sent in between
- At packet buffering level, the `Emission Buffer` and `Reception Buffer` handle CRC checking, the related ACK/Retry mechanism, and output queuing

Output queuing is defined as a Python script. It supports multiple priority levels, manages a dedicated queue of packets waiting for acknowledgement, and coordinates with the central switch for flow control. In the current version, we make simple choices:

- Priority levels are strict: there is no fair share discipline
- Each packet is ACK'ed individually: there is no group acknowledgement
- Packets to be re-emitted are put at the tail of the emission queue, like incoming packets
- When a queue is full, the port stops the main switch until a packet successfully leaves the queue

These choices can be flexibly changed in the Python script to search for better solutions. For example, priority queuing relies on per-priority queues; hence a filled-up, low-priority queue may block the main switch while taking time to drain. Improving that, while keeping the complexity low for efficient in-silicon implementation, is part of the challenges that our approach will help to address.

D. Adaptation Layer: Allowing Real Software in the Loop

The ERICA adaptation layer (see Figure 4) is composed of the following elements:

- The adaptation module integrated into DDS, which adapts its behaviour according to the underlying network
- A set of Ptolemy II extensions, namely:
    - ◦ `EricaAppPart`, which listens to packets from DDS, tags the data with their metadata – one of them being the application ID – then sends them to `ERICA Mapper` (and vice versa)
    - ◦ `ERICA Mapper`, making the bridge between simulated RapidIO nodes and applications
    - ◦ `EricaHostPart`, filtering packets from `ERICA Mapper` which relate to the node

We will now describe these components in more detail, and provide illustrations.

1) DDS Adaptation Module: The adaptation module we introduced in section III-A is the cornerstone of the ERICA architecture. It enables the use of DDS middleware over either a Ptolemy II simulated network or the SANDRA platform. It could eventually be extended to work over other networks – note that DDS already works over UDP/IP through its base implementation.

At first, working with Ptolemy II will allow us to test out how the DDS QoS parameters are being translated on networks of increasing complexity. Since the locations of the applications on the network are important and must be taken into account, it is very useful to be able to create different topologies over which our adaptation layer will be tested out. Ultimately, this will allow us to stress our fully simulated network with a real middleware behaviour that could neither be accurately nor simply simulated. As we explained in the introduction to this paper, bridging the simulated and real-world networks under a common application layer also makes it easier to focus on them, whatever happens above the adaptation module.

We have integrated the adaptation module within Vortex OpenSplice DDS, at the step where DDS packets are about to be sent out through a UDP socket. At this step, the module is basically intercepting the data buffers, IP adresses and ports according to the underlying network. If the underlying network is Ethernet, the base behaviour is retained, i.e. sending the packet through the socket. If the underlying network is the SANDRA platform (i.e. an in-silicon RapidIO node), addresses are translated to RapidIO addresses and the sending function from the SANDRA API is called. If working over a Ptolemy II simulated RapidIO network, the DDS adaptation module wraps the data with associated metadata (application ID, node ID, destination ID) and sends them to a UDP listener at the Ptolemy II level.

2) Ptolemy II Extensions for ERICA: The main idea is to use Ptolemy II publish-subscribe actors to map applications to their corresponding node in the simulated network. Publish-subscribe actors are useful for connecting applications to their corresponding nodes, without having to physically draw any

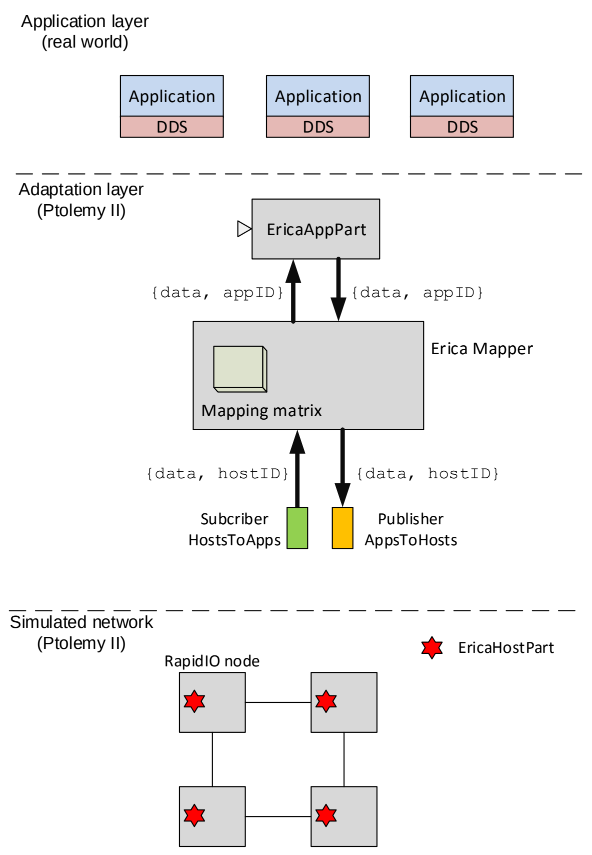


Fig. 8. Overall architecture of the Ptolemy II adaptation layer

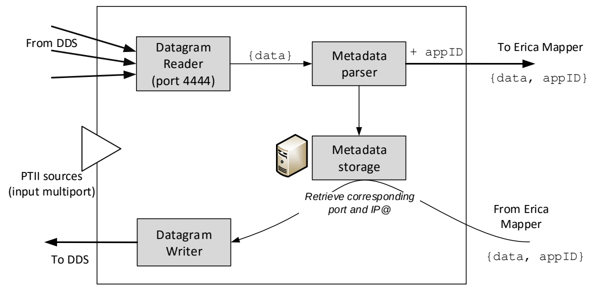


Fig. 9. EricaAppPart model

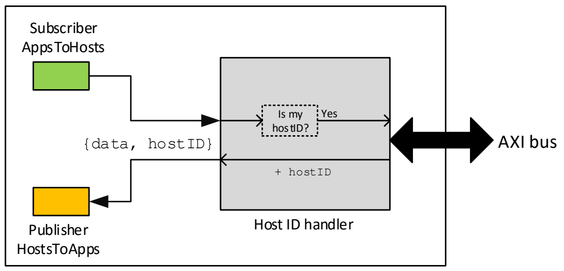


Fig. 10. EricaHostPart model

actor link. This allows us to separate the application and network layers and centralize the mapping into a unique matrix, thanks to the design described in Figure 8.

All applications are connected to all nodes through two pub-sub channels that go from RapidIO nodes to `ERICA Mapper` and vice versa.

The `EricaAppPart` actor (Figure 9) listens to packets from DDS, parses the metadata of incoming packets, and then emulates the sending of the corresponding RapidIO packet from the right node, via `ERICA Mapper`. Conversely, when receiving data from the simulated network, it rebuilds the IP packet expected by the upper DDS layers and forwards it via UDP to the application. This leaves the `EricaAppPart` fully transparent from the DDS point of view. `EricaAppPart` also contains an input multiport into which simulated applications can be plugged, allowing mixing of real and simulated applications.

The `ERICA Mapper` actor is composed of a mapping matrix and a script:

- Downstream, it tags the data from the applications with the target host ID and publishes it to the nodes. Although each node then receives these data, they all discard them, except for the node with the corresponding host ID. This filtering is done thanks to the `EricaHostPart` actor (see Figure 10).
- Upstream, the RapidIO node tags the data with his host ID and publishes it to `ERICA Mapper`, thanks to `EricaHostPart`. The `ERICA Mapper` component then looks into the mapping matrix and forwards the data to the right application, by tagging them with the target application ID.

Using this technique, the application set and the network are almost separated: changing the underlying topology or the application set only involves editing the mapping matrix without drawing or deleting any actor link.

3) Wrap-up: With our architecture, the application and DDS layers may address either UDP/IP or our network, simulated or in-silicon. Conversely, either these applications, or simulated ones (based on Ptolemy II source actors) may address our simulator. However, although this architecture does work with one application and one node, it does not work with multiple applications and nodes, for a simple reason: Ptolemy II does not allow the creation of more than one `Publisher` for a given channel, because it would make the model non-deterministic. We thus have to adapt this base design, by creating dedicated channels per node (Figure 11). We are willing to find a better solution than having to manage so many channels, however. Except for this issue, Ptolemy II has shown very acceptable performances for the simulation of RapidIO with our model, given the scenarios we fed it with.

To summarize, our architecture allows real and simulated components to work together, and to be switched in a flexible way. This is required in order to experiment with routing

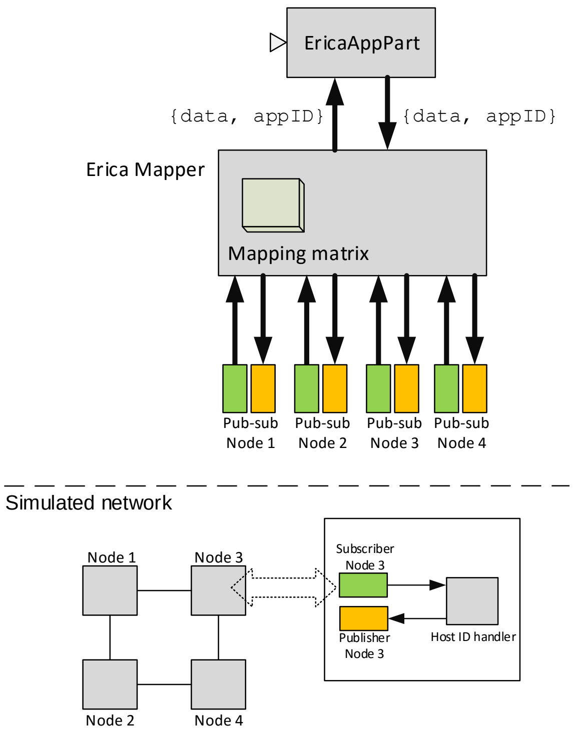


Fig. 11. Current architecture of the Ptolemy II adaptation layer

policies on a step-by-step basis, from full simulation to a real application on a physical network.

IV. Experiments and Results

Once the simulator behaviour has been validated with respect to the RapidIO specification, we have to tune the latencies and verify that end-to-end latencies are consistent with those measured on the platform. For this purpose, we firstly use simulated applications (i.e. Ptolemy II source actors) in order to ensure very simple, harnessed communications.

A. *First Step: Overall Checking of the Model*

The first scenario is the basic point-to-point sending of a 4-byte payload, RAM-to-RAM. The goal is to validate our *sequential sections* and make sure that the overall simulation architecture is accurate enough with respect to the platform, for this simple scenario. We measure, on the platform, the values of latencies for each sequential section, then inject them into the simulation model, at the places shown in Figure 6. We then measure the end-to-end (i.e. RAM-to-RAM) latency on the simulator and the platform in order to compare them. Although we could just calculate this latency – by summing values – to make sure that our theorical model is good compared to the platform, measuring it from the simulator allows us to assess the actual model in the same process. Measurements have been performed three times, and average values are given in Table II.

As we can see, there is a huge gap between end-to-end latencies, although each latency section has been tuned with exact values. This gap is unacceptable because it would mean that the majority of significant latencies has been neglected, making our simulation not sufficiently reliable. This issue has two potential causes: either we have to take other latencies into account, that we did not think of; or there is an issue with the platform implementation, leading to the appearance of an unexpected latency. After investigating this issue, we find that a data width conversion FIFO (32 bits to 64 bits) is the source of the problem. This 64-bit bus is used to feed 4 transceivers when running in 4x mode with 16 bits of data per clock cycle. The FIFO always waited to be filled to 64 bits before forwarding the data. In our case, since there was only one packet to be sent, the End-of-Packet Control Symbol was stuck inside it. This would have happened each time the transferred data were not aligned to a 64-bit boundary. After having removed this latency, the measured end-to-end latencies are those given in Table III.

TABLE II. Measured latencies for sending of a simple 4-byte payload

Sequential section	Latency (ns)
AXI read	296 ns
Switching delay	56 ns
Emission Buffer	124 ns
Emission Wire	285 ns
Reception Wire	355 ns
Reception Buffer	124 ns
Switching delay (2)	56 ns
AXI-SRIO Bridge	168 ns
AXI write	226 ns
Σ	1,690 ns
RAM-to-RAM on PTII	**1,690 ns**
RAM-to-RAM on SANDRA	**5,100 ns**
Δ	**67%**

TABLE III. Measured end-to-end latencies after the implementation bug fix

RAM-to-RAM on PTII	1,690 ns
RAM-to-RAM on SANDRA	1,625 ns
Δ	**4%**

For this scenario, we showed that our model is accurate enough in terms of latencies, even if the end-to-end value from the simulator is slightly higher than that measured on the platform. This is due to measurement uncertainties, notably for RAM latencies, whose values may significantly vary. As previously explained, we are not aiming to compute Worst Case Transmission Time, but only a sufficiently accurate transmission time.

This first scenario shows the benefit of the symbiotic approach of ERICA, which is to cross-check values from the simulator and the in-silicon platform. This approach helped us to find a major issue within the implementation that we would probably not easily have found otherwise, considering

TABLE IV. MEASURED LATENCIES FOR THE EMISSION BUFFER SEQUENTIAL SECTION AND DEDUCED AFFINE FUNCTION

Packet size (bytes)	Latency (ns)
4 B	124 ns
32 B	232 ns
150 B	711 ns
248 B	1,096 ns
Deduced function	$\boldsymbol{L(S) = 4.00 \times S + 107}$ where $L(S)$ is the latency for an S-byte payload in nanoseconds

TABLE V. END-TO-END LATENCIES FOR DIFFERENT PAYLOAD SIZES

Packet size (bytes)	On PTII	On SANDRA
32 B	2,222 ns	2,099 ns
100 B	4,221 ns	4,113 ns
150 B	5,512 ns	5,436 ns
248 B	8,248 ns	8,226 ns
$\overline{\boldsymbol{\Delta}}$	**2.5%**	

the fact that the latency is very low even with this huge relative increase.

B. Second Step: Tuning Latencies

Although the latencies configured within Ptolemy II have led us to a sufficiently accurate end-to-end latency, they are tuned using values corresponding to a fixed-size payload. Although this first step was mandatory in order to check that the overall model was consistent with the SANDRA platform – and it also led us to a major fix for the latter –, but we aim to propose a model that is able to compute accurate latencies, irrespective of the payload size. Latencies in some sequential sections are constant, while in others they increase in linear fashion, according to the payload size. For each *sequential section*, we thus measure the corresponding latencies for payloads of multiple sizes. We then deduce the affine functions to compute latencies in the simulator with respect to the payload sizes. Since there are some inaccuracies in measurements, values do not increase in a perfectly linear way. As a consequence, we use simple linear regression to find the best approximated functions. An example is given in Table IV for the `Emission Buffer` sequential section.

Once we have tuned each sequential section in this way, we assess this computation model by cross-checking end-to-end latencies with the SANDRA platform. We use the same scenario as above (one sending of a packet), but with different payload sizes. The ensuing results are given in Table V.

This validates the reliability of our method for building an adaptive though very simple latency computation model.

C. Third Step: Stressed Node

Modelling latencies through several sections will allow us to maintain accurate latencies irrespective of the complexity of the scenario, because, as stated above, each sequential section corresponds to a defined part of the network behaviour.

TABLE VI. END-TO-END LATENCY FOR THE LAST OF 6 × 240-BYTE PAYLOADS SENT SIMULTANEOUSLY

	Latency (ns)
Calculated	20,311 ns
Measured via Ptolemy II	20,311 ns
Measured via the VHDL simulator	19,332 ns
$\boldsymbol{\Delta}$	**5%**

In a given scenario, each global latency thus results from a combination of sequential sections, taking into account the payload sizes and intermediary queues.

We take as an example the sending of six 240-byte payloads at the same time. We assume, given the payloads size and our previous measurements, that the bottleneck is located at the `AXIread` sequential section. The end-to-end latency for the last payload should then be:

$$L(N) = L_1 + CSS(S) \times (N-1)$$
$$L(6) = L_1 + AXIr(240) \times 5$$
$$L(6) = 20,311\, ns$$

where:

$L(N)$ is the end-to-end latency for the Nth payload,

L_1 is the end-to-end latency for the first payload,

$CSS(S)$ is the latency corresponding to the Critical Sequential Section, i.e. the sequential section where the bottleneck is located, given the payload sizes,

$AXIr(S)$ is the latency corresponding to the `AXIread` sequential section, for an S-byte payload.

We measure this latency on our simulator and the SANDRA VHDL simulator. We have to use this cycle-accurate simulator because we are not yet able to run this scenario on the platform. We obtain the values given in Table VI. The results show the benefit of our architecture, which is still able to compute an accurate end-to-end latency when the scenario is more complex. Of course, this experiment is not sufficient to strictly validate the model irrespective of the scenario, but it shows the flexibility and scalability of our approach.

V. DISCUSSION AND ONGOING WORK

Although we have built several scenarios and cross-checked the simulation model with the SANDRA implementation, the current state of the latter prevents us from assessing the error management feature, i.e. the sending of Packet-Not-Accepted Control Symbols in case of a corrupted packet. We thus plan to develop an error injection feature and build more complex scenarios, e.g. communication via several intermediary nodes or a more stressed network. The presented work helped us to assess our overall model and the benefit of our symbiotic approach.

As we explained in the introduction, it is important that the application clock is consistent with that of the simulator. We are currently developing a lightweight solution based on open-source virtualization components. This work is not completely finished, but has already shown promising results.

As for the SANDRA platform, we will implement RapidIO DMA logical layer support and integrate it into the DDS ERICA module (see III.D). This DMA logical layer fits our needs better than the messaging one we are currently using.

In the longer term, we aim to design an overall software that makes it easier to conduct experiments with ERICA, for instance by automatically deploying the applications on the SANDRA platform or by generating a Ptolemy II model from a higher level topology drawing. This would also allow us to build virtual networks and machines to logically separate applications that are mapped with different RapidIO nodes, and achieve time consistency.

With regard to the dynamic mesh networking project, research will now focus on real-time QoS cross-layer support, i.e. the design of low-level switching and dynamic scheduling mechanisms and their linking to DDS QoS policies. This work will be done thanks to ERICA.

VI. Conclusion

We are addressing embedded real-time systems whose communications between components tend to become more and more demanding and complex. This reflects the increasing weight of embedded applications in terms of computation and communication requirements. We consider mesh networking of sensors and computing units to be an elegant solution to reconcile real-time constraints with needs in terms of throughput and dynamicity. However, we need a way of enforcing guaranteed real-time QoS from the highest layers down to the mesh network configuration and behaviour. In this paper, we have presented our symbiotic approach to designing this QoS support via a discrete-event Ptolemy II simulated network that is connected with a real implementation of DDS. We have also described how an in-silicon Serial RapidIO platform can help us to define a more representative network model, while the latter may highlight bugs within the implementation. This symbiotic approach enhances our experimentation process and makes it more effective by bridging the simulated and physical worlds. Finally, we discussed the limitations of our approach and presented some ongoing work.

References

[1] RapidIO Trade Association, "RapidIO Specification 3.1," Oct. 2014. [Online]. Available: http://www.rapidio.org

[2] S. Fuller, *RapidIO: The Embedded System Interconnect*. John y & Sons, Jan. 2005.

[3] A. Benveniste, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *6th Int. Symp. Formal Methods for Components and Objects*, 2008, pp. 200–225.

[4] "FRESCOR: Framework for Real-time Embedded Systems based on COntRacts," Tech. Rep. [Online]. Available: http://www.frescor.org

[5] Object Management Group, "Data Distribution Service (DDS), Version 1.4," Apr. 2014. [Online]. Available: http://www.omg.org/spec/DDS/

[6] [Online]. Available: http://prismtech.com/vortex/vortex-opensplice

[7] [Online]. Available: http://ptolemy.eecs.berkeley.edu/ptolemyII/

[8] R. M. Fujimoto, "Parallel and distributed simulation systems," Dec. 2001.

[9] "IEEE Standard for Modeling and Simulation, High Level Architecture (HLA) – Framework and Rules," *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pp. 1–38, Aug. 2010.

[10] K. Perumalla, "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances," in *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, Dec. 2006, pp. 84–95.

[11] S. Strassburger, T. Schulze, and R. Fujimoto, "Future trends in distributed simulation and distributed virtual environments: Results of a peer study," in *Winter Simulation Conference*, Dec. 2008, pp. 777–785.

[12] J. Dahmann and K. Morse, "High Level Architecture for simulation: an update," in *2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications, 1998. Proceedings*, Jul. 1998, pp. 32–40.

[13] C. D. Carothers, R. M. Fujimoto, R. M. Weatherly, and A. L. Wilson, "Design and Implementation of HLA Time Management in the RTI Version F.0," in *Proceedings of the 29th Conference on Winter Simulation*, ser. WSC. IEEE Computer Society, 1997, pp. 373–380. [Online]. Available: http://dx.doi.org/10.1145/268437.268511

[14] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, "Distributed Simulation of Heterogeneous and Real-Time Systems," in *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Oct. 2013, pp. 55–62.

[15] S. Taylor, "HLA-CSPIF The High Level Architecture COTS Simulation Package Interoperability Forum," in *Proceedings of the 2003 Fall Simulation Interoperability Workshop (WCS)*, 2003.

[16] S. Taylor, X. Wang, S. Turner, and M. Low, "Integrating heterogeneous distributed COTS discrete-event simulation packages: an emerging standards-based approach," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 36, no. 1, pp. 109–122, Jan. 2006.

[17] B. Camus, C. Bourjot, and V. Chevrier, "Combining DEVS with Multi-agent Concepts to Design and Simulate Multi-models of Complex Systems," Jan. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01103892

[18] B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, 2nd ed. Orlando, FL, USA: Academic Press, Inc., 2000.

[19] J. Vaubourg, Y. Presse, B. Camus, C. Bourjot, L. Ciarletta, V. Chevrier, J.-P. Tavella, and H. Morais, "Multi-agent Multi-Model Simulation of Smart Grids in the MS4SG Project," in *PAAMS'15*, ser. Lecture Notes in Computer Science, vol. 9086, Jun. 2015, p. 12. [Online]. Available: https://hal.inria.fr/hal-01171428

[20] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings of the 31th Conference on Local Computer Networks*, ser. LCN '06. Tampa, FL, USA: IEEE Computer Society, 2006, pp. 641–648.

[21] H. Aydt, S. Turner, W. Cai, and M. Low, "Research issues in symbiotic simulation," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, Dec. 2009, pp. 1213–1222.

[22] [Online]. Available: http://www.opencores.com/project,rio

[23] Bombardier, "Bombardier." [Online]. Available: http://www.bombardier.com/

[24] Xilinx, "Xilinx Zynq." [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[25] ——, "Xilinx." [Online]. Available: http://www.xilinx.com/legal.htm

[26] [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[27] ARM, "ARM Trademark." [Online]. Available: http://www.arm.com/about/trademarks/

[28] ——, "ARM AMBA Trademark." [Online]. Available: http://www.arm.com/about/trademarks/arm-trademark-list/AMBA-trademark.php

[29] J. Kurose and H. Mouftah, "Computer-aided modeling, analysis, and design of communication networks," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 1, pp. 130–145, Jan. 1988.

[30] H. Mouftah and R. Sturgeon, "Distributed discrete event simulation for communication networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 9, pp. 1723–1734, 1990.

Session 7

Code Generation

Wednesday 27th, 14:45 – Ariane 1

Fine-Tuning the Accuracy of Numerical Computations in Avionics Automatic Code Generators

Alexis Werey[†,*], David Delmas[*] and Matthieu Martel[†]

[*]Airbus Operations SAS, Toulouse, France
[†]University of Perpignan, Laboratoire de Mathématiques et Physique LAMPS, France

Abstract

Most of safety-critical embedded software, such as fly-by-wire control programs, performs a lot of floating-point computations. High level specifications are expressed in a formal model edited manually in SCADE through a graphical interface. It generally handles numerical variables and constants as if they were ideal real numbers. This work, for the purpose of numerical accuracy analysis, presents a new version of an Automatic Code Generator (ACG). This tool transforms high-level models into C codes and performs static computations by using multiple-precision arithmetic. This article describes a successful way of controlling computation accuracy of numerical constants in an Automatic Code Generator. An accuracy analysis on numerical constant values is presented in a case study.

Keywords: automatic code generator, industrial software, floating-point computation, numerical accuracy, constant propagation

1 Introduction

In this article, we focus on a method for tuning the accuracy of numerical computations in an avionic Automatic Code Generator (ACG), i.e. a software tool which aims at transforming a formal model into a target source code (generally in C). The generated codes are dedicated to be embedded into avionic systems. Nowadays, the accuracy of computations in avionic systems depends on floating-point arithmetic. Our motivation is to study the enhancement of floating-point computations in these codes by applying program transformation [11, 12]. For this purpose and as a first step, numerical error analysis is needed on static computation of constants during the code generation. Some constants are computed from the internal formal model by the ACG for code efficiency reasons due to CPU consumption. The analysis is performed by : a) observing algorithms accuracy manually or using static analysis by abstract interpretation [21, 20] ; b) checking systematically the accuracy of the resulting floating-point values after generation.

Assurance The embedded software are categorized at a Design Assurance Level depending of the safety impact on the system, in the sense of the avionic standard DO-178C/ED-12C [1]. In fact, the ACG has to be qualified as a development tool at the same assurance level as the generated software.

Numerical optimization In order to lower the number of computations and accept almost only arithmetic operations in the embedded code, the ACG provides simplified constants. A similar situation in compilers arises when static expressions are computed at compile-time (e.g `const y=1;x=y+1` replaced by `x=2` by the compiler.) The fact that almost any compiler makes the same processings than our ACG that makes the results presented in this article rather general.

Floating-point arithmetic errors The floating-point representation necessary leads to numerical errors created by roundings and propa-

gated by operations. In practice, they are highly negligible compared with the uncertainty of the input data (precision of the sensors/actuators). However as part of a long-term perspective, the evolution of the hardware precision and the sophistication of control algorithms could necessitate in the future a better accuracy concerning the computation of numerical operations.
We aim at verifying that a combination of constants of the high level model is not likely to degrade the quality of calculations. Indeed, constants are often called several times inside a loop. The objective in the present work is to make use of the state of the art to generate constants as accurate as possible and then to be able to measuring floating-point errors. A tunable approach using the rational representation [8] for arithmetic operations $(+, -, \times, \div)$ and the extended-precision [9] floating-point representation for elementary functions (such as $\sqrt{}$, tan) has been selected for the ACG reimplementation.

This article is organized as follows : in Section 2, we give an overview of the high critical real time software code generation. Section 3 introduces the floating-point arithmetic and its use for constant propagation. Then Section 4 describes the details of the reimplementation. Section 5 presents experimental results on a use case. Finally, Section 6 gives general perspectives and Section 7 concludes.

2 Embedded Code Generation

In this section, we describe briefly the development of embedded software products. Figure 2 illustrates the development process.

2.1 Model Specification

In the context of Airbus, the high-level formal specifications of avionics program behaviours are expressed by SCADE sheets [2] and translated into the Lustre [3] synchronous data-flow programming language.
The implementation of this specification is processed automatically through an Automatic Code Generator developed internally.
The model representation consists in data (real, integer or boolean as either constants or variables) and data-flow structures (symbols and their inputs/outputs). Examples of symbols include delays and cosines, as depicted in Figure 1.

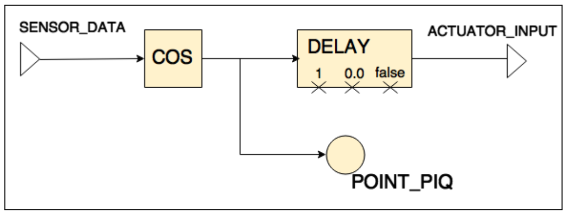


Figure 1 – An example of a SCADE sheet

Symbols may possibly hold constant parameters, e.g. the *cos* symbol in the Figure 1 does not contain constants while the *delay* symbol has 3 constants. Some other constants are global and not directly referenced in the sheet such as the clock.

2.2 Generated Code Structure

A symbol library is implemented by hand in C or assembly using macro-functions. The generated code, which has the same semantics as the initial formal specification, can therefore call these predefined macros. For hardware efficiency and safety reasons, macros are written as simple as possible with IEEE754 [4] well-defined operations with the double precision format. Macros and specification nodes can in addition differ on the number and the type of constant parameters since the ACG reduces the CPU-time by evaluating static expressions during the code generation.
A static expression in a symbol can be computed once for all by the ACG from either input and global constants in the model or by simplification of an algorithm thanks to mathematical properties. This method is similar, from a certain point of view, to the constant propagation optimization performed by compilers [5].

2.3 Numerical Error Analysis

In the process of qualification, the ACG is subject to analysis and in particular to floating-point accuracy. Note that during the step of optimization that we describe in the next section, the ACG computes values in floating-point double precision. Two complementary sorts of analysis are carried out :

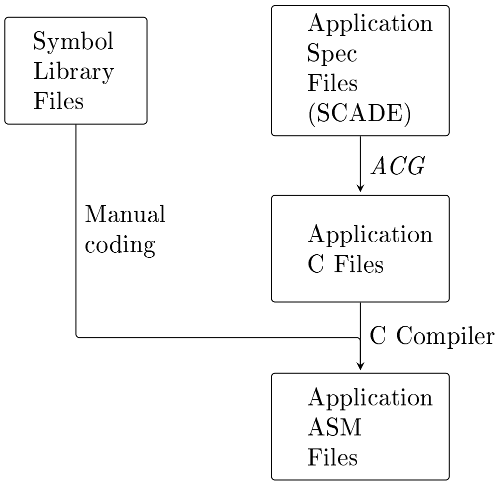


Figure 2 – The C development chain of the software

- A handwritten analysis followed by a static analysis [21] on floating-point accuracy to over-approximately estimate the errors generated by the algorithm.

- A set of tests to verify whether the ACG is near or far from the expected results.

Our objective is to provide a method to automatically analyse the current ACG with inputs from the formal model.
Note that some tools are already used for the analysis of control-command programs. *Astre* computes safe bounds on floating-point variables, *Fluctuat* computes error bounds between the exact and the floating-point semantic, both by abstract interpretation. *Compcert* is a certified compiler which use Coq libraries to handle the floating-point arithmetic [17].

3 Constant propagation of floating point expressions

3.1 The IEEE754 Standard

The IEEE-754 standard [4] describes the floating-point number format as well as the elementary operations $(+, -, \times, \div)$ and the square root function. It defines:

1. the simple precision on 32 bits with 8 bits for the exponent e, 23 bits for the mantissa m and 1 bit for the sign s

2. the double precision on 64 bits with 11 bits for the exponent, 52 bits for the mantissa and 1 bit for the sign

3. the double extended precision on a number of bits higher than 79 with, more than 15 bits for the exponent, more than 63 bits for the mantissa and 1 bit for the sign

It also specifies several kinds of floating-point numbers, e.g. in radix 2:

- normalized numbers represented by: $f = s \cdot (1.m_0...m_{p-1}) \cdot 2^e$ where $s \in \{-1, 1\}$, p the length of m and $E_{min} \leq e \leq E_{max}$ ($E_{min} = -1022$ and $E_{max} = 1023$ in double precision)

- denormalized numbers defined when $e = 0$ and represented by: $f = s \cdot (0.m_0...m_{p-1}) \cdot 2^{E_{min}}$

- signed zeros $+0$ and -0

- infinities $\pm\infty$

- NaN (Not A Number)

The standard also defines 4 rounding modes : rounding $\circ_\sim$ to nearest, rounding $\circ_{-\infty}$ towards $-\infty$, rounding $\circ_{+\infty}$ towards $+\infty$ and rounding $\circ_0$ towards zero. Arithmetical operations are exactly rounded, which means that assuming $\uparrow_\circ$: $\mathbb{R} \to \mathbb{F}$ the function returning the floating-point value c a real number with the chosen rounding mode $\circ \in \{\circ\ , \circ_{-\infty}, \circ_{+\infty}, \circ_0\}$, and an arithmetical operation $\diamond \in \{+, -, \times, \div\}$, then for all floating-point numbers $f_1, f_2 \in \mathbb{F}$:

$$f_1 \diamond_{\mathbb{F},\circ} f_2 = \uparrow_\circ (f_1 \diamond_{\mathbb{R}} f_2) \tag{1}$$

Despite this propriety, floating-point operations can lead to subtle traps as in logical as in material considerations [6, 10]. Among several reasons of numerical errors, we can cite : the representation errors, for example the rational number $\frac{1}{10}$ is not representable in floating-point radix 2 ; the accumulation and propagation of errors through operations ; the absorption (resp. cancellation), a loss of precision when adding a number with a smaller one from a different order (resp. when subtracting two numbers approximately equal) ; unstable tests, the path executed in the control flow is different from the one expected in the real semantics.

3.2 Constant propagation

In the long term, we aim at improving the floating-point accuracy of the generated embedded code by reducing error propagation. A first step consists in analyzing the floating-point accuracy of the computation of constants during the code generation. Equation (1) gives an example of a first order low-pass filter algorithm, considering that $X(t)$ and $Y(t)$ are temporal values denoting respectively the input and the output at time t. The constants a, b and c are related to physical properties.

$$Y(t) = \frac{\frac{2b}{a}}{\frac{2b}{a}+1}\, Y(t-1) + \frac{1}{\frac{2b}{a}+1}\, (X(t)+X(t-1)) \quad (2)$$

The terms $c_1 = \dfrac{1}{1+\frac{2*b}{a}}$ and $c_2 = 1 - c_1$ can be calculated beforehand. The generated code then looks like:

$$Y(t) = c_2\; Y(t-1) + c_1\; (X(t) + X(t-1)) \quad (3)$$

The generated constants c_1 and c_2 appear several times in a function executed at every tick of a synchronous clock, their numerical accuracy is then worth to be considered. In addition, the embedded software environment constraints, for example the real time constraints, e.g. *Worst-Case Execution Time* (WCET) [7], are relevant at execution-time rather than at compile-time. In order to improve their accuracy, the floating-point operations can be computed by an arbitrary precision library such as the *Multi-Precision Floating-Point* library (*MPFR*) [9], which has in addition the benefit to be independent from the host machine. Indeed, static computations achieved by ACG, or more generally by compilers may be sensitive to the arithmetic of the processor as well to the dynamic libraries of the host machine and libraries get rid of this issue.

Example of floating-point rounding errors A first example of floating-point error that may arise is a call to the floor function (returning an integer) after some floating-point operations. If the floating-point arithmetic double precision is used to implement the real value $\lfloor\, x \times 1000 \rfloor$ when $x \in \mathbb{R}$ and $\lfloor . \rfloor : \mathbb{R} \to \mathbb{Z}$, the natural way is to write $floor(x \times 1000)$ when this time x is a floating-point variable, $\times$ is the standard floating-point multiplication and $floor$ the floating-point floor function returning an integer. Some values are error-prone, for $x = 1.001$, this implementation leads to an important relative error : 10^{-3}.

$$\begin{aligned} floor(1.001 * 1000) &= floor(1.000999999 * 1000) \\ &= floor(1000.999999) \\ &= 1000 \end{aligned} \quad (4)$$

3.3 Static Analysis

In this section, we illustrate the loss of accuracy resulting from the computation of constant parameters by a static analysis on the computation of the second order low-pass filter constants.

Low-pass filter transfer function

$$H(s) = \frac{1}{1 + 2\xi\frac{s}{\omega_0} + \frac{s^2}{\omega_0^2}}$$

This function specified in the formal model is implemented in a way that requires the following constant a :

$$a = \frac{\frac{2}{\tan^2(\omega_0 \frac{\Delta_t}{2})} - 2}{1 + \frac{2\xi}{\tan(\omega_0\frac{\Delta_t}{2})} + \frac{1}{\tan^2(\omega_0\frac{\Delta_t}{2})}} \quad (5)$$

Assuming that the clock Δ_t is 0.01 seconds, the analysis of a with *Fluctuat* and 1000 global subdivisions on ω_0 returns the results depicted in Figure 3.

The range of a is determined by the analysis from the input range of Δ_t, ω_0 and ξ and the relative error is the comparison of the real result and the floating-point one. Considering the errors, the generated floating-point constant a is sound and complying with the specifications.
Assuming that Δ_t is 0.02 seconds, Figure 4 show the relative error on the computation of a. Lower and upper bound of the relative error are drawn in the graph.

Constant	Range	Relative error
Δ_t	$[1.00000000 \cdot 10^{-2}; 1.00000001 \cdot 10^{-2}]$	$[-2.16840435 \cdot 10^{-17}; -2.03287906 \cdot 10^{-17}]$
ω_0	$[6.28318530 \cdot 10^{-1}; 9.42477797 \cdot 10^{1}]$	$[0;0]$
ξ	$[9.99999998 \cdot 10^{-2}; 1.20000001]$	$[0;0]$
a	$[5.95590847 \cdot 10^{-1}; 2.11425866]$	$[-6.52831414 \cdot 10^{-15}; 6.61998150 \cdot 10^{-15}]$

Figure 3 – Results of the static analysis from Equation 4

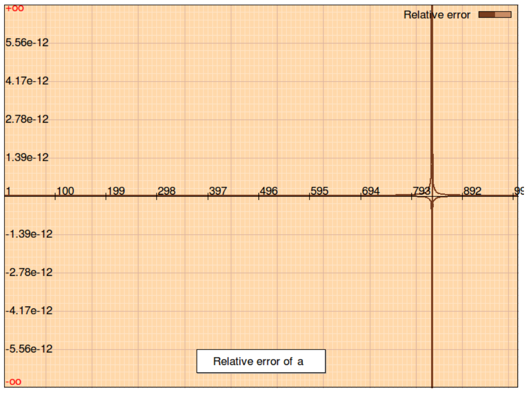


Figure 4 – Relative error bounds on a

Even though the absolute error stays stable, the relative error increases when $\omega_0 \simeq \frac{\pi}{2\Delta_t}$. A new implementation of the ACG would be an additive analysis tool to reinforce the assurance that constants are generated accurately.

4 A Tunable Code Generator

4.1 Alternative Arithmetics

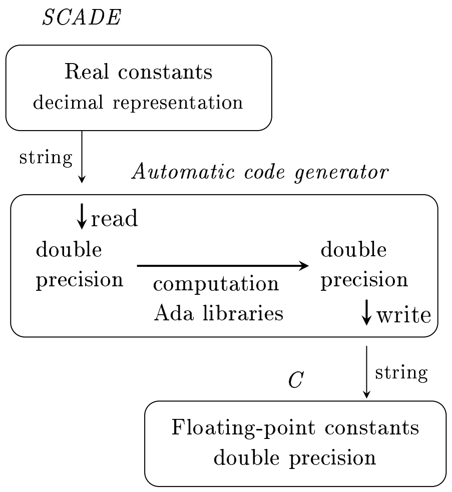
Several works has been done to estimate and to improve the numerical accuracy of programs [16]. We can cite interval arithmetic which consists of bounding the exact result by two floating-point numbers. Stochastic arithmetic [15] that consists of running the programs several times with random rounding modes. Doing so, the round-off errors are randomly propagated and the output is eventually statistically approximated.

The ones that would be considered for the reimplementation of the computations in the ACG is the rational and the extended-precision arithmetic. The rational representation is defined by two unbounded integers, a numerator and a denominator. The computations in this representation is exact and thus not subjected to any round-off errors, but valid only for the elementary operations $\{+, -, \times, \div\}$. Since the constants of the model are expressed in decimal and most of the operations in the ACG are elementary, this arithmetic is well-suited to this representation. The downside of this approach is that it is not stable in term of memory and time costs. Basically, after a certain number of operations on a variable x, the cost of a new operation is getting higher and higher due to the possible increasing length of the numerator and the denominator. However in our context, it does not have to be taken into account since most of the time few operations are involved in static computation of constants and not performed at run time.

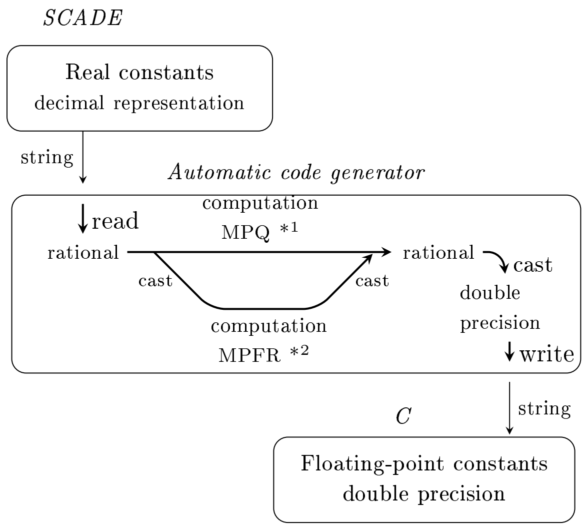
In some circumstances, for example on a call to a tangent or a square root function, the floating-point representation with extended precision is more appropriate and can be configured with the selected number of bits.

4.2 Architectures

We reimplement the existing ACG written in Ada to improve the numerical accuracy of the constant computations. The current ACG uses the double precision complying with the precision of the IEEE754 double format. As shown in Figure 5a, the library Ada is used to compute the static expressions and both the reading and the writing process of constants generate rounding errors. In addition, there is some call to the tangent function which is not defined in the IEEE754 Standard but which is defined in the dynamic mathematical library of the host machine. Figure 5b describes the computation process of the new ACG using the multi-precision rational arithmetic library (MPQ) from GMP [8] and the *MPFR* library.



(a) Static computation in the current ACG



(b) Static computation in the optimized ACG
$*^1$: only with arithmetic operators
$*^2$: with elementary functions

5 Experimental Results

In order to have representative results of static computations, the former and the new ACG have been tested on a use case representative of the model for a large avionic system (several thousand nodes). They have been executed on a Sun Solaris platform on a SPARC architecture. MPFR has been configured to the to-nearest rounding mode and 200 bits of allocation for a floating-point number. We compare by analysis of the generated constants the numerical errors created by the ACGs. The analysis calculates for each constant the relative error between the two values. If c_1 is generated by the first ACG and c_2 by the second from the same static expression, the relative error corresponds to $\frac{|c_1-c_2|}{c_2}$ as we suppose that the second ACG computes almost exactly. Figure 6 depicts the results. N indicates the number of generated constants whose relative error from the analysis is greater than the error fixed in x-axis. For clarity, Figure 7 shows the index of the most significant bit in the mantissa of the absolute error $|c_1 - c_2|$ between the related constants c_1 and c_2 for the 1000 worst cases. Some conclusions have been raised from this experiment : The results are complying with the accuracy requirements for the former ACG. There are approximately 1000 constants whose relative error is above 10^{-10} and 500 above 10^{-9} on a total of 40000 generated constants. They all are in conformance with the internal error criterion expressed by an acceptable error bound.

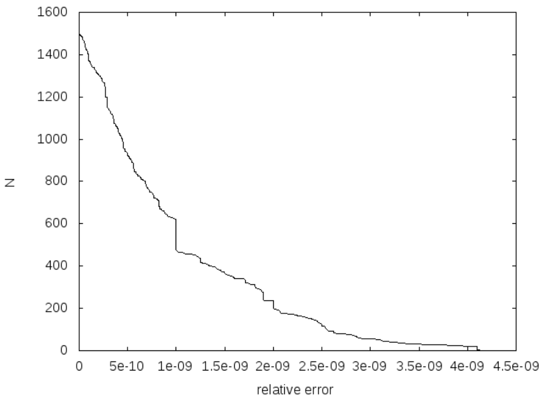


Figure 6 – Relative error between generated constants

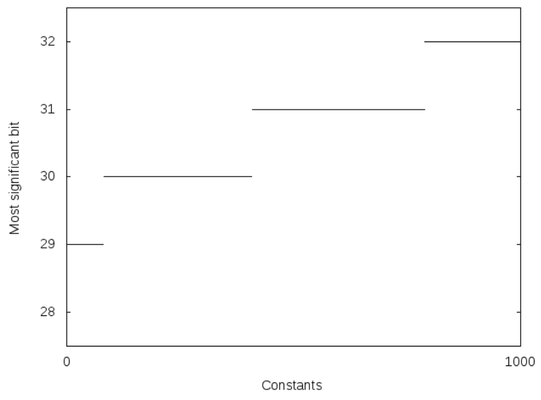


Figure 7 – Index of the most significant bit of the absolute error

6 Perspectives

The work introduced in this article, concerning the accurate evaluation of the static arithmetic expressions by ACGs, is a first step towards the automatic generation of fully optimized code with respect to the accuracy of floating-point computations. The next steps are the generation of accurate arithmetic expressions and, more generally, the generation of accurate programs. More precisely, in the floating-point arithmetic, the accuracy of expressions depends on how formulas are written and mathematically equivalent expressions give different results, more or less accurate, when evaluated by the machine. For example, $a+(b+c)$ is generally different from $(a+b)+c$ and $x+x^2$ is generally different from $x(x+1)$. The choice of the best formula depends on the ranges of the inputs which can be obtained by static analysis [20].
Automatic techniques enabling one to find a very accurate implementation of a formula for the IEEE754 arithmetic have been introduced in [12] and other work has been done to re-organize larger pieces of code containing several statements made of assignments, conditionals and loops [13]. For example, `x=a+b ; y=c+d; z=x+y` may be rewritten as `z=((a+c)+d))+b` if this choice is relevant, depending on the ranges of `a`, `b`, `c` and `d` computed by static analysis. These transformations still needs to be extended to the inter-procedural case. At mid-term, ACGs could take advantage of these techniques to generate accurate code, made of formulas mathematically equivalent to the ones of the high level model (e.g. *Scade*) but taylored to evaluate very accurately in the computer arithmetic.
The code transformations operated to improve the numerical accuracy may lead to programs which are different enough from the original ones while they still computing the same mathematical results. Another important research direction is to take care of certification. Traceability and a good level of confidence in the transformed codes are obviously mandatory. We aim at generating correctness certificates ensuring that the transformed codes are mathematically equivalent and more accurate than the original codes. These certificates will be expressed using formal proof assistants. In practice, we plan to use the Coq theorem prover. A related problem, more theoretical is to show that substituting a more accurate piece of code to an original piece of code inside a large program improves the accuracy of the whole application. Ongoing research in currently done in this direction.

7 Conclusion

In this article, we have shown how the static computations performed by ACGs are sensitive to the host machine on which the code generation is performed. Indeed, the resulting code may depend on the arithmetic of the host machine as well as on the dynamic libraries of its operating system. This problem is not limited to ACGs. It is general to all the modern compilers which perform constant propagation during their optimization passes. Our experimental results show that large industrial applications may be impacted by this transformation. In addition, even if the accuracy of the computations done at compile-time without using any specific high precision library like *MPFR* is acceptable, reproducibility and maintenance questions still remain since compiling again the same program on another machine, possibly several years later to deliver a new version of the software, may yield a program embedding constants which are different from these of the original code.
The accuracy tunable ACG is intended to be used as an analysis tool and may be subject to a process of qualification with the corresponding libraries *GMP MPQ* and *MPFR* in the case of a development application.

Another aspect of the problem of improving the accuracy concerns the execution-time. We wish the transformed codes, optimized for accuracy, be at least as efficient as the original codes. The transformations introduced in [12, 13, 14] neither slow the applications nor speed them up significantly. However, the transformation of the arithmetic expression is only guided by accuracy. The existing methods could be interestingly extended to search a compromise between time and accuracy, or, in other word, a rewriting of the computations which improves both accuracy and execution-time even if it not optimal for each criterion taken separately.

References

[1] DO-178C/ED-12C, Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE, December 2011.

[2] F.-X. Dormoy. Scade 6 a model based solution for safety critical software development. In Embedded Real-Time Systems Conference, 2008.

[3] N. Halbwachs, P. Caspi and D. Pilaud. The synchronous dataflow programming language Lustre. Another Look at Real Time Programming, Proceedings of the IEEE, Special Issue, Sept. 1991.

[4] ANSI/IEEE, IEEE Standard for Binary Floating-point Arithmetic, Std 754-2008, ANSI/IEEE, 2008.

[5] G. A. Kildall, A Unified Approach to Global Program Optimization, 1973.

[6] D. Monniaux, The Pitfalls of Verifying Floating-Point Computations, ACM, 2008.

[7] J. Souyris, V. Wiels, D. Delmas, H. Delseny. Formal Verification of Avionics Software Product, 2009.

[8] T. Granlund and the GMP development team. The GNU Multiple Precision Arithmetic Library, http://gmplib.org/, 2012.

[9] L. Fousse, G. Hanrot, V. Lefèvre, Patrick Pélissier and Paul Zimmermann. A Multiple-Precision Binary Floating-Point Library with Correct Rounding, ACM Transactions on Mathematical Software 2007.

[10] D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM Comput. Surv., 1991.

[11] M. Martel, Semantics-Based Transformation of Arithmetic Expressions, 14th International Symposium, SAS 2007.

[12] A. Ioualalen and M. Martel. A New Abstract Domain for the Representation of Mathematically Equivalent Expressions, 19th International Symposium, SAS, 2012.

[13] N. Damouche, M. Martel and A. Chapoutot. Intra-procedural Optimization of the Numerical Accuracy of Programs, Formal Methods for Industrial Critical Systems, 20th International Workshop, 2015.

[14] P. Panchekha, Alex Sanchez-Stern, James R. Wilcox and Zachary Tatlock. Automatically improving accuracy for floating point expressions, Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 2015.

[15] F. Jézéquel and J. M. Chesneaux, CADNA: a library for estimating round-off error propagation, Computer Physics Communications, 2008.

[16] J. C. Bajard, O. Beaumont, J. M. Chesneaux, M. Daumas, J. Erhel, D. Michelucci, J. M. Muller, B. Philippe, N. Revol, J. L. Roche, J. Vignes. Qualité des calculs sur ordinateurs, vers des arithmétiques plus fiables ?, Masson, 1997.

[17] G. Melquiond, Floating-point arithmetic in the Coq system, Inf. Comput., 2012.

[18] S. Boldo, J. C. Filliâtre and G. Melquiond, Combining Coq and Gappa for Certifying Floating-Point Programs, Intelligent Computer Mathematics, 16th Symposium, MKM, 2009.

[19] Modern Compiler Implementation in C, Cambridge University Press, 1998.

[20] Static Analysis by Abstract Interpretation of Numerical Programs and Systems, and FLUCTUAT, Static Analysis - 20th International Symposium, SAS, 2013.

[21] David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal, and Franck Védrine. Towards an industrial use of fluctuat on safety-critical avionics software, FMICS, 2009.

CompCert – A Formally Verified Optimizing Compiler

Xavier Leroy[1], Sandrine Blazy[2],
Daniel Kästner[3], Bernhard Schommer[3], Markus Pister[3], Christian Ferdinand[3]

1: Inria Paris-Rocquencourt, domaine de Voluceau, 78153 Le Chesnay, France

2: University of Rennes 1 - IRISA, campus de Beaulieu, 35041 Rennes, France

3: AbsInt Angewandte Informatik GmbH. Science Park 1, D-66123 Saarbrücken, Germany

Abstract

CompCert is the first commercially available optimizing compiler that is formally verified, using machine-assisted mathematical proofs, to be exempt from miscompilation. The executable code it produces is proved to behave exactly as specified by the semantics of the source C program. This article gives an overview of the design of CompCert and its proof concept and then focuses on aspects relevant for industrial application. We briefly summarize practical experience and give an overview of recent CompCert development aiming at industrial usage. CompCert's intended use is the compilation of life-critical and mission-critical software meeting high levels of assurance. In this context tool qualification is of paramount importance. We summarize the confidence argument of CompCert and give an overview of relevant qualification strategies.

1 Introduction

Modern compilers are highly complex software systems that try to find a balance between various conflicting goals, like minimal size or minimal execution time of the generated code, maximum compilation speed, maximum retargetability, etc. The code generation process itself is a collection of complex transformation steps, many of which are essentially NP-complete, as e.g., the standard backend phases instruction selection, instruction scheduling, or register allocation. The quality of a compiler typically is rated on the efficiency of the generated code whereas its price should not be too high. In consequence compilers have to be efficiently developed, their structure is complex, they contain many highly tuned and sophisticated algorithms – and they can contain bugs. Studies like [12, 5] and [14] have found numerous bugs in all investigated open source and commercial compilers, including compiler crashes and miscompilation issues. Miscompilation means that the compiler silently generates incorrect machine code from a correct source program. Such wrong-code errors can be detected in the normal software testing stage which, however, does typically not include systematic checks for them. When they occur in the field, they can be hard to isolate and to fix.

Whereas in non-critical software functional software bugs tend to have bigger impact than miscompilation errors, the importance of the latter dramatically increases in safety-critical systems. Contemporary safety standards such as DO-178B/C, ISO-26262, or IEC-61508 require to identify potential functional and non-functional hazards and to demonstrate that the software does not violate the relevant safety goals. Many verification activities are performed at the architecture, model, or source code level, but all properties demonstrated there may not be satisfied at the executable code level when miscompilation happens. This is not only true for source code review but also for formal, tool-assisted verification methods such as static analyzers, deductive verifiers, and model checkers. Moreover, properties asserted by the operating system may be violated when its binary code contains wrong-code errors induced when compiling the OS. In consequence, miscompilation is a non-negligible risk that must be addressed by additional, difficult and costly verification activities such as more testing and more code reviews at the generated assembly code level.

The first attempts to formally prove the correctness of a compiler date back to the 1960's [10]. Since 2015, with the CompCert compiler, the first formally verified optimizing C compiler is commercially available. What sets CompCert apart from any other production compiler, is that it is formally verified, using machine-assisted mathematical proofs, to be exempt from miscompilation issues. In other words, the executable code it produces is proved to behave exactly as specified by the semantics of the source C program. This level of confidence in the correctness of the compilation process is unprecedented and contributes to meeting the highest levels of software assurance. In particular, using the CompCert C compiler is a natural complement to applying formal verification techniques (static analysis, program proof, model checking) at the source code level: the correctness proof of CompCert C guarantees that all safety properties verified on the source code automatically hold as well for the generated executable.

Usage of CompCert offers multiple benefits. First, the cost of finding and fixing compiler bugs and shipping the patch to customers can be avoided. The testing effort required to ascertain software properties at the binary executable level can be reduced. Whereas in the past for highly critical applications (e.g., according to DO-178B Level A) compiler optimizations were often completely switched off, using optimized code now becomes feasible.

The article is structured as follows: in Sec. 2 we give an overview of the CompCert design and illustrate its proof concept in Sec. 3. Sec. 4 summarizes experimental results and practical experience obtained with the CompCert compiler. Specific developments required for industrial application of CompCert are discussed in Sec. 5. Sec. 6 summarizes the confidence argument for CompCert and discusses tool qualification strategies, Sec. 7 concludes.

2 CompCert Structure

Like other compilers, CompCert is structured as a pipeline of compilation passes, depicted in Figure 1 along with the intermediate languages involved. The 20 passes bridge the gap between C source files and object code, going through 11 intermediate languages. The passes can be grouped in 4 successive phases, described next.

Parsing Phase 1 performs preprocessing (using an off-the-shelf preprocessor such as that of GCC), tokenization and parsing into an ambiguous abstract syntax tree (AST), and type-checking and scope resolution, obtaining a precise, unambiguous AST and producing error and warning messages as appropriate. The LR(1) parser is automatically generated from the grammar of the C language by the Menhir parser generator, along with a Coq proof of correctness of the parser [9]. Optionally, some features of C that are not handled by the verified front-end are implemented by source-to-source rewriting over the AST. For example, bit-fields in structures are transformed into regular fields plus bit shifting and masking. The subset of the C language handled here is very large, including all of MISRA-C 2004 [11] and almost all of ISO C99 [8], with the exceptions of variable-length arrays and unstructured, non-Misra `switch` statements (e.g. Duff's device).

C front-end compiler The second phase first rechecks the types inferred for expressions, then determines an evaluation order among the several permitted by the C standard. This is achieved by pulling side effects (assignments, function calls) outside of expressions, turning them into independent statements. Then, local variables of scalar types whose addresses are never taken (using the `&` operator) are identified and turned into temporary variables; all other local variables are allocated in the stack frame. Finally, all type-dependent behaviors of C (overloaded arithmetic operators, implicit conversions, layout of data structures) are made explicit through the insertion of explicit conversions and address computations. The front-end phase outputs Cminor code. Cminor is a simple, untyped intermediate language featuring both structured (`if`/`else`, loops) and unstructured control (`goto`).

Back-end compiler This third phase comprises 12 of the passes of CompCert, including all optimizations and most dependencies on the target architecture. It bridges the gap between the output of the front-end and the assembly code by progressively refining control (from structured control to control-flow graphs to labels and jumps) and function-local data (from temporary variables to hardware registers and stack frame slots). The most important optimization performed is register allocation, which uses the sophisticated Iterated Register Coalescing algorithm [7]. Other optimizations include function inlining, instruction selection, constant propagation, common subexpression elimination (CSE), and redundancy elimination. These optimizations implement several strategies to eliminate computations that are useless or redundant, or to turn them into equivalent but cheaper instruction sequences. Loop optimizations and instruction scheduling optimizations are not implemented yet.

Optimization passes exploit the results of two intraprocedural static analyses: a forward "value" analysis, that infers variation intervals for integer variables, known values for floating-point variables, and nonaliasing information for pointer variables; and a backward "neededness" analysis, generalizing liveness analysis by identifying the bits of a variable or memory location that do not contribute to the final results of a function.

Optimizations and static analyses are performed over the RTL intermediate representation, consisting of control-flow graphs of "three-address" machine-like instructions. RTL code is not in Single Static Assignment (SSA) form. Consequently, static analyses are more costly and optimizations slightly less aggressive than what is possible with SSA-based compilation algorithms. The reason for not using SSA is that, at the beginning of the CompCert project, the semantic properties of SSA and the soundness arguments for SSA-based algorithms were poorly understood. The recent work of Demange *et al.* [2] and Zhao *et al.* [15] provides formal foundations for SSA-based optimizations.

Assembling The final phase of CompCert takes the AST for assembly language produced by the back-end, prints it in concrete assembly syntax, adds DWARF debugging information coming from the parser (cf. Sec. 5),

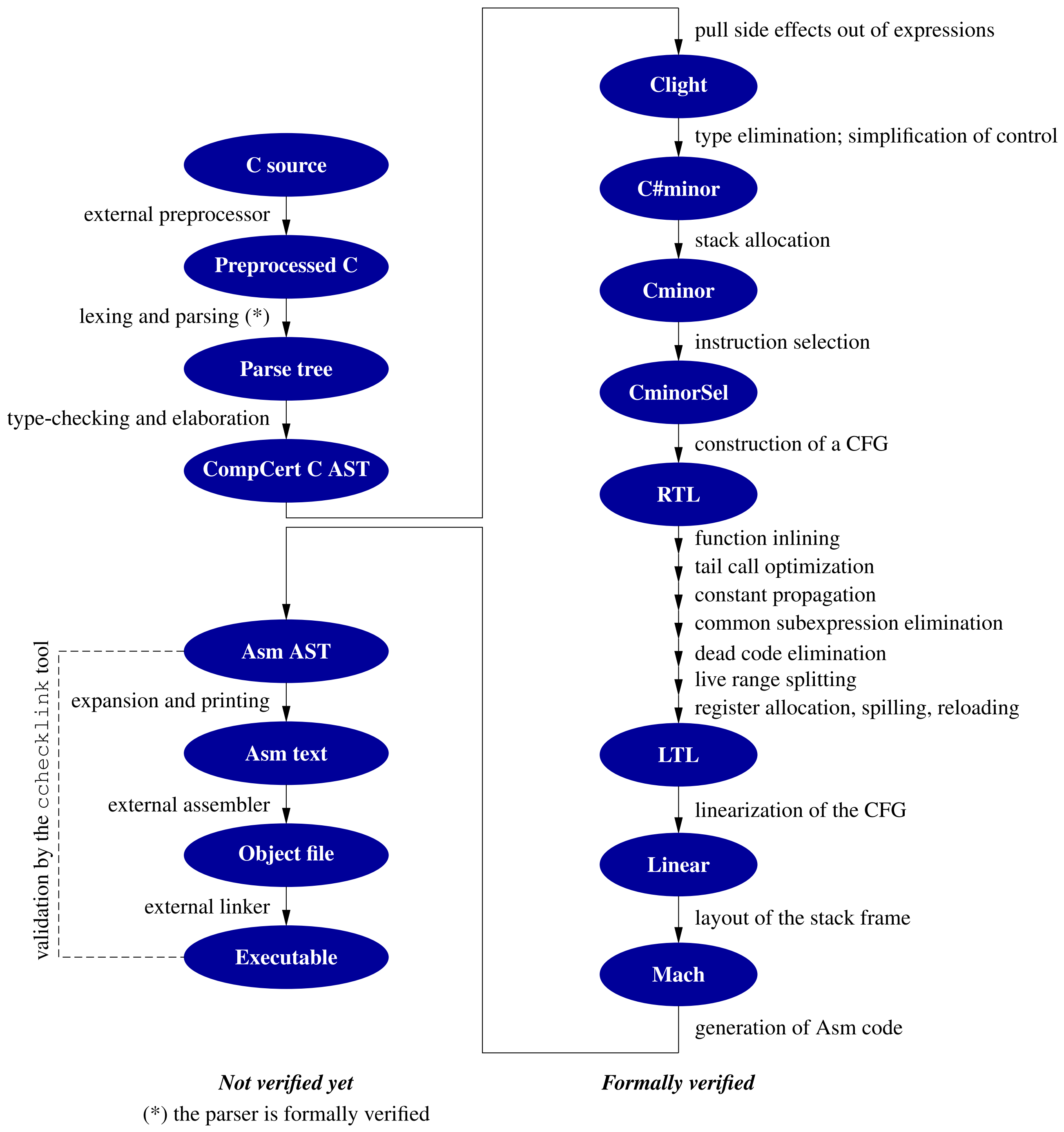


Figure 1: General structure of the CompCert C compiler

and calls into an off-the-shelf assembler and linker to produce object files and executable files. To improve confidence, CompCert provides an independent tool that re-checks the ELF executable file produced by the linker against the assembly language AST produced by the back-end.

3 The CompCert Proof

The CompCert front-end and back-end compilation passes are all formally proved to be free of miscompilation errors; as a consequence, so is their composition. The property that is formally verified is *semantic preservation* between the input code and output code of every pass. To state this property with mathematical precision, we give formal semantics for every source, intermediate and target language, from C to assembly. These semantics associate to each program the set of all its possible behaviors. Behaviors indicate whether the program terminates (normally by exiting or abnormally by causing a run-time error such as dereferencing the null pointer) or runs forever. Behaviors also contain a trace of all observable input/output actions performed by the program, such as system calls and accesses to "volatile" memory areas that could correspond to a memory-mapped I/O device. Arithmetic operations and non-volatile memory accesses are not observable.

Technically, the semantics of the various languages are specified in small-step operational style as labeled transition systems (LTS). A LTS is a mathematical relation *current state* $\xrightarrow{trace}$ *next state* that describes one step of execution of the program and its effect on the program state. For assembly-style languages, the state of the program comprises the values of processor registers and the contents of memory. The step of computation is the execution of the instruction pointed to by the program counter (PC) register, which updates the contents of registers and possibly of memory. For higher-level languages such as C, states have a richer structure, including not just memory contents and an abstract program point designating the statement or expression to execute next, but also environments mapping variables to memory locations, as well as an abstraction of the stack of function calls.

A generic construction defines the observable behaviors from these transition systems, by iterating transitions from an initial state (the initial call to the `main` function): $S_0 \xrightarrow{t_1} S_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} S_n$. Such sequences of transitions can stop on a state S_n from which no transition is possible. This describes a terminating execution, where the program terminates either normally (on returning from the `main` function) or on a run-time error (e.g. dereferencing the null pointer, or dividing by zero). Alternatively, an infinite sequence of transitions describes a program execution that runs forever. In both cases, the concatenation of the traces $t_1.t_2\ldots$ describes the I/O actions performed. Several behaviors are possible for the same program if non-determinism is involved. This can be internal non-determinism (e.g. multiple possible evaluation orders in C) or external non-determinism (e.g. reading from a memory-mapped device can produce multiple results depending on I/O behaviors).

To a first approximation, a compiler preserves semantics if the generated code has exactly the same set of observable behaviors as the source code (same termination properties, same I/O actions). This first approximation fails to account for two important degrees of freedom left to the compiler. First, the source program can have several possible behaviors: this is the case for C, which permits several evaluation orders for expressions. A compiler is allowed to reduce this non-determinism by picking one specific evaluation order. For example, consider the following C program:

```
#include <stdio.h>
int f() { printf("f"); return 1; }
int g() { printf("g"); return 2; }
int main() { return f() + g(); }
```

According to the C semantics, two behaviors are allowed, producing `fg` or `gf` as output, depending on whether the call to `f` or the call to `g` occurs first. CompCert chooses to call `g` first, hence the compiled code has only one behavior, with `gf` as output.

Second, a C compiler can "optimize away" run-time errors present in the source code, replacing them by any behavior of its choice. (This is the essence of the notion of "undefined behavior" in the ISO C standards.) Consider an out-of-bounds array access:

```
int main(void)
{ int t[2];
  t[2] = 1;  // out of bounds
  return 0;
}
```

This is undefined behavior according to ISO C, and a run-time error according to the formal semantics of CompCert C. The generated assembly code does not check array bounds and therefore writes 1 in a stack location. This location can be padding, in which case the compiled program terminates normally, or can contain the return address for "main", smashing the stack and causing execution to continue at PC 1, with unpredictable effects. Finally, an optimizing compiler like CompCert can notice that the assignment to `t[2]` is useless (the `t` array is not used afterwards) and remove it from the generated code, causing the compiled program to terminate normally.

To address the two degrees of flexibility mentioned above, CompCert's formal verification uses the following definition of semantic preservation, viewed as a refinement over observable behaviors:

> If the compiler produces compiled code C from source code S, without reporting compile-time errors, then every observable

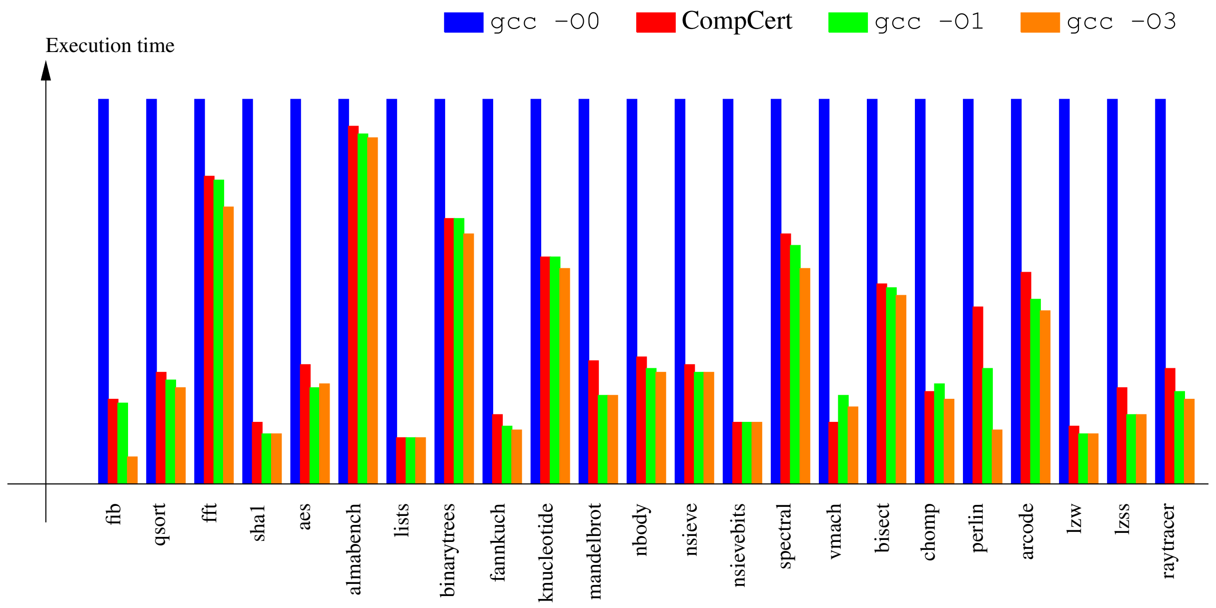


Figure 2: Performance of CompCert-generated code relative to GCC 4.1.2-generated code on a Power7 processor. Shorter is better. The baseline, in blue, is GCC without optimizations. CompCert is in red.

behavior of *C* is either identical to an allowed behavior of *S*, or improves over such an allowed behavior of *S* by replacing undefined behaviors with more defined behaviors.

In the case of CompCert, the property above is a corollary of a stronger property, called a simulation diagram, that relates the transitions that *C* can make with those that *S* can make. First, 15 such simulation diagrams are proved independently, one for each pass of the front-end and back-end compilers. Then, the diagrams are composed together, establishing semantic preservation for the whole compiler.

The proofs are very large, owing to the many passes and the many cases to be considered – too large to be carried using pencil and paper. We therefore use machine assistance in the form of the Coq proof assistant. Coq gives us means to write precise, unambiguous specifications; conduct proofs in interaction with the tool; and automatically re-check the proofs for soundness and completeness. We therefore achieve very high levels of confidence in the proof. At 100 000 lines of Coq and 6 person-years of effort, CompCert's proof is among the largest ever performed with a proof assistant.

4 Practical Experience

CompCert targets the following three architectures: 32-bit PowerPC, ARMv6 and above, and IA32 (i.e. Intel/AMD x86 in 32-bit mode with SSE2 extension). On PowerPC and ARM, the code generated by CompCert runs at least twice as fast as the code generated by GCC without optimizations, and approximately 10% slower than GCC 4 at optimization level 1, 15% slower at optimization level 2 and 20% slower at optimization level 3. These numbers were obtained on the benchmark suite shown in figure 2, along with the performance numbers for a Power7 processor. This suite comprises computational kernels from various application areas: signal processing, physical simulation, 3d graphics, text compression, cryptography. By lack of aggressive loop optimizations, performance is lower on HPC codes involving dense matrix computations. On IA32, due to its paucity of registers and its specific calling conventions, CompCert is approximately 20% slower than GCC 4 at optimization level 1.

CompCert provides a general mechanism to attach free-form annotations (text messages possibly mentioning the values of variables) to C program points, and have these annotations transported throughout compilation, all the way to the generated assembly code, where the variable names are expressed in terms of machine code addresses and machine registers. Apart from improving traceability this source annotation mechanism enables WCET tools to compute more precise WCET bounds. Indeed, WCET tools like aiT [6] operate directly on the executable code, but they sometimes require programmers to provide additional information (e.g., the bound of a while loop) that cannot easily be reconstructed from the machine code alone. When using CompCert, such information can be safely extracted from annotations inserted at the source code level. A tool automating this task was developed by Airbus: it generates a machine-level annotation file usable by the aiT WCET analyzer. Compiling a whole flight control software from Airbus (about 4 MB of assembly code) with CompCert resulted in significantly improved performance in terms of WCET bounds and code size [3].

5 Industrial Application

An industrial application of CompCert is not only attractive because of the high confidence in the correctness of the compilation process. As mentioned in Sec. 1 it opens up the possibility to use optimized code even in highly critical avionics projects, hence enabling higher software performance. Furthermore, CompCert's annotation mechanism as described in Sec. 4 can contribute to further improving confidence by providing proven traceability information [3].

A prerequisite for industrial use of a compiler is the possibility to debug the program under compilation. While previous versions of CompCert only provided rudimentary debugging support, CompCert now is able to produce debug information in the Dwarf 2 format. It generates debugging information for functions and variables, including information about their type, size, alignment and location. This also includes local variables so that the values of all variables can be inspected during program execution in a debugger. To this end CompCert introduces an additional pass which computes the live ranges of local variables and their locations throughout the live range. Furthermore CompCert keeps track of the lexical scopes in the original C program and creates corresponding Dwarf 2 lexical scopes in the debugging information.

The CompCert sources can be downloaded from Inria[1] free of charge; the current state of the development can be viewed on Github [2]. In addition, a revisioned distribution is available from AbsInt, either as a source code download or as pre-compiled binary for Windows and Linux. To create the pre-compiled binary, Coq is executed under Linux to create the OCaml source files and the corresponding correctness proof. This provides a proven-in-use argument for Coq usage since there is a wide community using Coq under Linux/Unix. Furthermore it makes sure the Windows and Linux versions operate on the same sources. Under Windows the OCaml sources are compiled with a native Windows compiler; to execute CompCert no additional libraries (e.g., cygwin or mingw) are needed. The package also contains pre-configured setup files for the compiler driver to control the cooperation between the CompCert executable and the external cross compiler required for preprocessing, assembling and linking.

Translation Validation. Currently the verified part of the compile tool chain ends at the generated assembly code. In order to bridge this gap we have developed a tool for automatic translation validation, called *Valex*, which validates the assembling and linking stages a posteriori.

Valex checks the correctness of the assembling and linking of a statically and fully linked executable file against the internal abstract assembly representation produced by CompCert. In order to use Valex, the C source files for the program must be compiled by CompCert and the command-line option `-sdump` must be specified. This option instructs CompCert to serialize the internal abstract assembly representation in JSON [4] format.

The generated `.json`-files as well as the linked executable are then passed as arguments to the Valex tool. The main goal is to verify that every function defined in a C source file compiled by CompCert and not optimized away by it can be found in the linked executable and that its disassembled machine instructions match the abstract assembly code. To that end, after parsing the abstract assembly code Valex extracts the symbol table and all sections from the linked executable. Then the functions contained in the abstract assembly code are disassembled. Extraction and disassembling is done by two invocations of exec2crl, the executable reader of aiT [1].

exec2crl tries to decode all symbols in the linked executable which have the same name as a function symbol contained in any of the input `.json`-files. If one of these symbols in the linked executable is not a function, exec2crl reports a warning that it cannot decode the given symbol. The decoded control-flow graph is linearized to a sequential list of its instructions sorted by their address.

The main stages of Valex deal with function and variable usage. For every function in the abstract assembly code, Valex matches the instructions in the abstract assembly code against the instructions contained in the linked executable. Valex supports both the cases that an abstract assembly instruction directly corresponds to one machine instruction, and that it corresponds to a sequence of machine instructions. It checks whether the arguments of the instructions are equivalent and the intended instruction mnemonic is used.

For every variable defined in a C source file compiled by CompCert Valex checks whether the corresponding symbol can be found in the symbol table. It also checks that the corresponding size and initialization data is contained in the linked executable and matches the initialization data in the abstract assembly. Valex reports an error if one of these checks fails.

In a further stage Valex reconstructs the mapping from symbolic names and labels to machine addresses computed by the linker, checks that there is an address for every symbolic name and label and ensures that all occurrences are always mapped to the same address.

Additionally, Valex tests for variables and functions whether they are placed in their corresponding sections, which are either the default sections, or the sections specified by the user in the C files using `#pragma` section. When using the GCC tool chain, Valex also checks whether uninitialized variables that were placed into one of CompCert's internal sections `Data`, `Small Data`,

[1]http://compcert.inria.fr/download.html
[2]https://github.com/AbsInt/CompCert

`Const` or `Small Const` are contained in the `.bss` or `.sbss` section of the executable.

Currently Valex can check linked PowerPC executables that have been produced from C source code by the CompCert C compiler using the Diab assembler and linker from Wind River Systems, or the GCC tool chain (version 4.8, together with GNU binutils 2.24).

6 The Confidence Argument

As described in Sec. 3 all of CompCert's front-end and back-end compilation passes are formally proved to be free of miscompilation errors. These formal proofs bring strong confidence in the correctness of the front-end and back-end parts of CompCert. These parts include all optimizations – which are particularly difficult to qualify by traditional methods – and most code generation algorithms.

The formal proofs do not cover the following aspects:

1. Correctness of the specifications used for the formal proof, i.e. the formal semantics of C and assembly.
2. The parsing phase, i.e. the transformation from the input C program to CompCert's abstract syntax.
3. The assembling and linking phase.

Those aspects can be handled well by traditional qualification methods, i.e. via a validation suite, to complement the formal proofs. A validation suite for CompCert is currently in development and will be available from AbsInt.

Especially the parsing phase (item 2) can be seen as a straightforward code generation pass which does not include any optimizations and only performs local transformations. Since the internal complexity of this stage is low, systematic testing provides good confidence. CompCert can print the result of parsing in concrete C syntax, facilitating comparison with the C source.

However, it is possible to provide additional confidence beyond the significance of the validation suite, in particular for items 1 and 3. CompCert provides a reference interpreter, proved correct in Coq, that can be used to systematically test the C semantics on which the compiler operates. Likewise, the Valex validator described in Sec. 5 provides confidence in the correctness of the assembling and linking phase. It performs translation validation of the generated code which is a widely accepted validation method [13].

At the highest assurance levels, qualification arguments may have to be provided for the tools that produce the executable CompCert compiler from its verified sources, namely the “extraction” mechanism of Coq, which produces OCaml code from the Coq development, combined with the OCaml compiler. We are currently experimenting with an alternate execution path for CompCert that relies on Coq's built-in program execution facilities, bypassing extraction and OCaml compilation. This alternate path runs CompCert much more slowly than the normal path, but fast enough that it can be used as a validator for selected runs of normal CompCert executions.

In summary, CompCert provides unprecedented confidence in the correctness of the compilation phase: the 'normal' level of confidence is reached by providing a validation suite, which is currently accepted best practice; the formal proofs provide much higher levels of confidence concerning the correctness of optimizations and code generation strategies; finally, the Valex translation validator provides additional confidence in the correctness of the assembling and linking stages.

7 Conclusions

CompCert is a formally verified optimizing C compiler: the executable code it produces is proved to behave exactly as specified by the semantics of the source C program. Experimental studies and practical experience demonstrate that it generates efficient and compact code. Further requirements for industrial application, notably the availability of debug information, and support for Linux and Windows platforms have been established. Explicit traceability mechanisms enable a seamless mapping from source code properties to properties of the executable object code. We have summarized the confidence argument for CompCert, which makes it uniquely well-suited for highly critical applications.

References

[1] AbsInt GmbH, Saarbrücken, Germany. *AbsInt Advanced Analyzer for PowerPC*, October 2015. User Documentation.

[2] G. Barthe, D. Demange, and D. Pichardie. Formal verification of an SSA-based middle-end for CompCert. *ACM Trans. Program. Lang. Syst.*, 36(1):4, 2014.

[3] R. Bedin França, S. Blazy, D. Favre-Felix, X. Leroy, M. Pantel, and J. Souyris. Formally verified optimizing compilation in ACG-based flight control software. In *ERTS 2012: Embedded Real Time Software and Systems*, 2012.

[4] ECMA International. Standard ECMA-404 The JSON Data Interchange Format. Technical report, ECMA International, Oct. 2013.

[5] E. Eide and J. Regehr. Volatiles are miscompiled, and what to do about it. In *EMSOFT '08*, pages 255–264. ACM, 2008.

[6] C. Ferdinand and R. Heckmann. aiT: Worst-Case Execution Time Prediction by Static Programm Analysis. In R. Jacquart, editor, *Building the Information Society. IFIP 18th World Computer Congress*, pages 377–384. Kluwer, 2004.

[7] L. George and A. W. Appel. Iterated register coalescing. *ACM Trans. Prog. Lang. Syst.*, 18(3):300–324, 1996.

[8] ISO. International standard ISO/IEC 9899:1999, Programming languages – C, 1999.

[9] J.-H. Jourdan, F. Pottier, and X. Leroy. Validating LR(1) parsers. In *ESOP 2012: 21st European Symposium on Programming*, volume 7211 of *LNCS*, pages 397–416. Springer, 2012.

[10] J. McCarthy and J. Painter. Correctness of a compiler for arithmetic expressions. In *Mathematical Aspects of Computer Science*, volume 19, pages 33–41, 1967.

[11] Motor Industry Software Reliability Association. MISRA-C: 2004 – Guidelines for the use of the C language in critical systems, 2004.

[12] NULLSTONE Corporation. NULLSTONE for C. `http://www.nullstone.com/htmls/ns-c.htm`, 2007.

[13] A. Pnueli, M. Siegel, and E. Singerman. Translation validation. In *TACAS'98: Tools and Algorithms for Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 151–166. Springer, 1998.

[14] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. In *PLDI '11*, pages 283–294. ACM, 2011.

[15] J. Zhao, S. Nagarakatte, M. M. K. Martin, and S. Zdancewic. Formal verification of SSA-based optimizations for LLVM. In *PLDI'13: ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 175–186. ACM, 2013.

Integration of Polychrony and QGen Model Compiler

Christophe Junke, Thierry Gautier, Jean-Pierre Talpin
INRIA, Rennes-Bretagne-Atlantique Research Centre, France
firstname.lastname@inria.fr

Loïc Besnard
CNRS, IRISA, Rennes, France
loic.besnard@irisa.fr

Abstract **We present the development of a model transformation tool between the synchronous Signal language and QGen model compiler's intermediate language as well as an alternative block sequencing implementation for QGen which supports strict partial and total orders. We discuss our contributions and their possible applications in the field of reactive system design as well as some experiments**

1 Introduction

The work presented in this paper is about providing a semantic bridge between the data-flow language Signal and a model compiler named QGen developed by Adacore. QGen is the result of a collaborative work made in a project funded by the French FUI funding framework[1], named P, which was initiated to continue the work made in the Geneauto project [15]. Signal [3, 1] is a synchronous data-flow language for designing reactive systems. The language manipulates clocks and dependencies to perform a static scheduling of computations and data exchanges. Our objective in the P project was to use the Polychrony toolset of the Signal language to compute fined-grained static scheduling of computations and communications for P models based on architectural properties, as demonstrated in a previous work [19]. In this paper, we show a translation scheme between the formalism used by QGen, named P, and the Signal metamodel, SSME[2].

1.1 P project and QGen

The goal of the P project was to develop a qualifiable model compiler in the context of critical systems, which would group several existing heterogenous modelling formalisms under a single language, referred to as P language. The set of input languages envisioned for P was ranging from architecture description languages, such as AADL [11], SysML [13], Marte [21], to control and command languages such as Matlab Simulink [8] and SCADE [10]. The motivation behind the P project and in particular its goals regarding qualification is explained in details in [4, 5].

QGen is the compiler resulting from this combined effort and its first version was released February, 2015. It can import a subset of discrete Simulink models (models with a fixed time-step) and produce MISRA C [20] or Ada Ravenscar [6] code along with traceability information and formal annotations. The P language is layered as multiple languages describing different views of a same system. (i) *Code Model* is a model representation of imperative statements and functions. This language is an intermediate target before code optimizations and the actual code generation to C and Ada. (ii) *System Model* is inspired by Simulink and other block diagram formalisms, where systems are organized as data-flow graphs, connecting blocks by their data and control ports through signals. (iii) *Block library Model* is a configuration language describing block types. Block types are identified by names (e.g. "Gain", "Product") and can contain additional parameters. For example, the Sum block has different meaning depending on the number and type of its arguments (scalars, vectors, matrices). The QGen compiler is responsible for translating block instances as Code Model elements.

Typically, a model is first designed as a System Model and is gradually refined upto a point where imperative Code Model elements are generated and optimized. The QGen compiler is organized as multiple optional steps of transformations. The first one is a preprocessing step which removes organizational and visual features: for example, from/goto blocks, used for simplifying data-flow routing in a model, can be replaced by actual signals; also, artificial boundaries defined by virtual subsystems are removed, so that blocks previously grouped in a virtual system are moved as direct elements of the removed system's parent. Another interesting step is block sequencing. As a hierarchical data-flow graph, the System Model defines a partial order among blocks, which must be respected at execution time. The purpose of the block sequencer is to compute a sequential execution order for all blocks in a system according to data-flow dependencies and other attributes. Finally, a compilation step transforms System Model elements into Code Model elements. Without getting into details, each system is implemented as a function manipulating global variables, whereas blocks of a system are run in accordance to the total execution order computed at the previous step. Nested subsystems are implemented as function calls in the enclosing system's associated body. At any stage of the processing, the current model can be exported as XMI. Various options modify the behavior of the whole compilation process.

1.2 Signal and Polychrony

Signal is a synchronous data-flow language for designing reactive systems, based on infinite discrete flows of values and events called *signals*. A signal x represents an unbounded series $(x_t)_{t \in \mathbb{N}}$ of values at each logical instant t. At any instant, a signal may be present, at which point it holds a value, or absent, which is denoted by $\perp$. Signal values can be constructed from previously known values using a delay operator denoted $\$$ which shifts values of a signal by a fixed number of steps.

For each signal s is defined a clock $\hat{s}$ which can be rep-

[1] Fonds Unique Interministériel, http://competitivite.gouv.fr/
[2] Signal's Syntactic Model under Eclipse

resented as a boolean variable in the domain of clocks. A clock $\hat{s}$ denotes the set of logical instants at which the signal s has a value. The null clock $\hat{0}$ is always absent. Signal programs are described as *processes* (written here P or Q) which define relations between clocks and values of *signals* through signal equations. $(P \mid Q)$ denotes the synchronous composition of processes P and Q where equations represented by both P and Q are simultaneously considered: it corresponds to a union of constraints. The process P/x restricts the lexical scope of signal x to the process P.

Signal provides polychronous operators that are interpreted as clock constraints on signals, such as *when* or *default*. For example, *a when b* is an expression over signals whose clock is $\hat{a} \wedge \hat{b}$: this boolean formula means that the signal *a when b* is present exactly when both signals a and b are present. Similarly, *a default b* has the clock $\hat{a} \vee \hat{b}$, which means that, if during the execution of system being designed, either signal a or b is present, the signal *a default b* is also present. Signals having equivalent clocks are said to be synchronous and can be combined with monochronous operators, which include numerical operators (e.g. $a + b$) and spatial operators (e.g. array access).

A clock $\hat{d}$ is said to be a child clock of $\hat{c}$ if $\hat{d} \rightarrow \hat{c}$: whenever $\hat{d}$ is present, its parent clock $\hat{c}$ is also necessarily present. The signal compiler organizes clocks into a hierarchy during a static analysis called *clock calculus*. The compiler ensures that a signal is computed only when its clock can be determined to be true: it is the case if the clock is associated with a concrete boolean signal which is already known to be present and whose value is *true* at a particular reactive step. But it is also possible to infer that a clock variable is true based on other values and clocks, thanks to clock constraints. If there are enough constraints on clocks to generate a deterministic reactive system where clocks are always known to be either present or absent at runtime the system is said to be endochronous.

Apart from clocks, Signal also allows to declare dependencies between signals: *{a –> b} when c* is a conditioned dependency that is active only when the formula $\hat{a} \wedge \hat{b} \wedge \hat{c} \wedge c$ is true: (i) clocks $\hat{a}$, $\hat{b}$ and $\hat{c}$ must be present, and (ii) the guard c itself must be true. When this relation holds, the value of b cannot be computed before the value of a is computed first.

The Polychrony toolset[3] provides a model-driven environment for the Signal language, including in particular a compiler, a front-end in the Eclipse platform[4], a model-checker for formal verification and other translators. The Signal compiler organizes computations as a conditional directed acyclic graph where arcs are labelled with clock expressions. Clocks and dependencies allow to easily distribute computations across multiple processing units (e.g. threads), while preventing deadlocks and using a miminum set of communications between units: since clock presence or absence can often be inferred from exchanged values, it is not always necessary to reify them as concrete boolean signals.

[3]`http://polychrony.inria.fr`

[4]Open-source framework POP with the Polarsys Industry Working Group: `https://www.polarsys.org/projects/polarsys.pop`

1.3 Outline

In section 2, we present our work on an unambiguous static block scheduler for QGen which can compute both partial and total orders based on user preferences. This scheduler was developed to help QGen interoperate with other tools which work on data-flow graphs, like Signal. In section 3, we describe our transformation function from the P language to SSME. In section 4, we discuss some experimental results with respect to our initial objectives.

2 Block sequencing

In the System Model subset of the P formalism, blocks represent computation nodes with input and output ports and signals describe values flowing between ports. Data-flow graphs encode a partial order of block executions: block sequencing is the action of finding a strict total order that refines the partial one. A strict total order is required when compiling reactive systems into sequential code. A compiler that would produce parallel sequences of code would not necessarily have to produce a total order, as long as dependencies are satisfied. A block sequencer must also ensure that circular dependencies are detected and should reject malformed models. A formally verified sequencer was implemented for the previous Geneauto [14] project but could not be used in QGen, which provides its own sequencer.

In the particular case of QGen, System Models are inspired by the Simulink language and as such, systems are hierarchical and may be given user-defined priorities (integers). Moreover, blocks are not necessarily always activated but subject to activation condition and control signals. QGen's earliest version provided a block sequencer that we will call here sequencer O ("original"). We will also describe our alternative sequencer implementation called N ("new" sequencer). Even though sequencer N is not integrated as-is into the current version of QGen, most of the problems found and reported to the development team of QGen while working on sequencer N are now addressed in sequencer O.

2.1 System Model language

2.1.1 Blocks and ports

A system in P is a functional view of a model represented by a hierarchical data-flow graph of blocks linked by directed connections called signals. *Blocks* can be either *system* blocks, *interface* blocks or *elementary* blocks. Blocks have input and output *ports*, which may be either *data* or *control* ports. *Signals* represent an ordered pair of ports. All blocks are named and typed according to an external predefined set of block definitions called a *Block library*. Block types are represented by a string and accept generic key/value parameters which are understood by the compiler to generate Code Models elements. *Elementary* blocks provide the actual computations and are implementation-defined. *Interface* blocks are used to represent the "port-block" family of blocks: for each input and output data or control port of a system, there is a *block* which represents that port inside that system. For example, an *Inport block* I_n inside a system S is an *interface* block having exactly one output *port* $o(I_n)$ which represents the value received from the input port $in(S)_n$ of S. *Interface* blocks differ from *elementary* blocks in that they hold an attribute called *portReference* which represents the interface port associated with the block. Also, interface blocks implementw some of the ports's semantics: the value of output ports can be *held* or *reset* when a port has been deactivated; *enabling* input control ports are used to trigger events when the input value is either raising, falling or changing.

2.1.2 Atomic systems

A *system* block is a container for nested blocks, and is assumed to be *atomic* in this section (we assume that models we consider have been preprocessed): (i) an atomic system can be computed only when all of its inputs are available; (ii) blocks that rely on outputs of an atomic system can be computed only when all the outputs of the atomic system have been computed; (iii) finally, while an atomic block is being computed, computation must not exit this block until all outputs are computed. See for example the blocks in figure 1: $i1$, $i2$ and $i3$ (resp. $o1$, $o2$ and $o3$) are three inputs (resp. outputs) of current subsystem. Blocks A, B and C are elementary blocks (interface blocks are not represented). Both blocks A and B are grouped into an atomic subsystem we call here S. Simple arrows represent actual signals whereas bold arrows which form crosses represent the dependencies implied by atomicity. The property (iii) above means that it is not possible to compute C between the computations of A and B, even though there is no dependency required by signals. In other words, the sequence (A, C, B) is not a compatible execution order for those three blocks. We cannot express atomicity with static partial dependencies because both (A, B, C) and (C, A, B) are compatible execution orders.

2.1.3 Datatypes

The P language defines a hierarchy of values to represent types in the target language of a compiled model. Types are either *primitive*, *array* or *pointer* types. Arrays represent multi-dimensional arrays and are composed of a base type and a list of dimensions. Among primitive types, there is a class of *numeric* types as well as *boolean*, *string*, *void* and *custom* types. Numeric types are divided into *complex* and *real* types (but complex are currently not used in QGen), where *real* types include both *integer*, *fixed-point* and either *single* or *double* floating-point values. Numeric types are described by a boolean value which indicates signedness (returned by *isSigned()*) as well as a width representing the number of bits available for this type (returned by *getNBits()*). Custom types designate external types. The QGen compiler provides C and Ada definitions for expected types, such as *GAUINT8* for unsigned 8-bits integers, as well as wrappers around mathematical and logical operators for those types.

2.2 Partial ordering of blocks

We defined the sequencer N to help QGen fulfill its interoperability objectives. QGen is expected to fit into existing methodological processes, which explains the strong emphasis put on separating concerns in the compiler: this can be witnessed by the existence of compiler flags for choosing

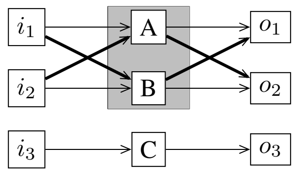


Figure 1: dependencies implied by atomicity of system block (in bold)

which transformations are applied on a model, as well as for exporting it as an XMI document between steps. In particular, it can be desirable to delegate block sequencing to an external tool, such as SynDEX [12, 17] or Polychrony. The reason is that block sequencing is a type of static scheduling of resources, which can be optimized by specialized tools, especially with distributed code.

A requirement of this approach is that the external tool must provide an ordering that is *compatible* with QGen's ordering of blocks. However, this requires to define a compatibility criterion. For example, is the total order provided by an external tool said to be compatible if that tool ignores user-defined priorities? Similarly, let us consider control signals (see section 2.4.2): QGen's compiler considers that a block B triggered by a control signal emitted from block A is implemented as-if B's code was inlined inside the body of A's code. That semantics leads to additional dependencies which could be seen as artificial for other tools but should be respected when working with QGen.

In order to satisfy different interpretations of what is the minimal set of dependencies the sequencer should consider, we let the user provide a *sorting policy* list made of predefined sort criteria. Those criteria are names of refinement steps to be applied on the dependency graph being built. For example, the DF criterion introduces data-flow dependencies implied by data-flow signals, whereas UP represents those implied by user-defined priorities. The TR criterion forces dependencies of triggered blocks to be dependencies of calling blocks (recursively): this criterion is applied whenever the model is to be compiled by QGen, since it represents the specific way the tool deals with block activations, but is optional because one might want to give a model to another tool. Some criteria represent actions to perform during block sequencing: *LOAD* reads all the dependencies currently stored in a model and introduces them in the dependency graph stored in memory. Another criterion is *ENSURE_TOTAL*, which makes the sequencer abort if that graph still represents a partial order at the moment the criterion is applied. Also, we proposed to modify the existing sequencer so that it can produce either a partial or total order, based on those input parameters.

With this approach, we can avoid the problem of having to implement multiple block sequencers and we can apply the following methodology when working with external toolchains: for some input model M_1, run QGen's sequencer with a given sorting policy P_1, which results in a partial order, exported in model M_2. Pass M_2 to some external tool, which can refine block ordering and produce model M_3. In order to check that M_3 has a compatible order with M_2, run QGen on model M_3 with a modified policy P_2 defined as P_1 followed by a LOAD operation and possibly other refinement steps. Incompatible orderings are then defined as those which introduce circular dependencies in a model. They are detected during the LOAD operation, which could introduce errors if other tools did not respect the original partial order.

This approach needs to be able to represent partial orders in P models. This is done by adding a block attribute named *nextExecutableBlocks*, the list of all blocks that depend on a particular block. We also tried to adapt the original sequencer O so that it could compute partial orders, but there were shortcomings with the existing approach that made it easier to write a more general sequencer N. Sequencer N, unlike O, is split into two parts (two Ada packages): (i) a dedicated yet generic graph data-structure representing partial and total orders and (ii) the actual sequencer which translates blocks and their attributes as a graph while processing the sorting policy given by the user. We detail these in the following sections.

2.3 Graph data-structure

The original sequencer O used to compute dependencies by assigning a rank to blocks. More precisely, blocks with no predecessors are assigned rank 0, and blocks for which all their predecessors have a rank are assigned a rank $M + 1$, where M is the maximal rank of those predecessors. This first step admitted a simple implementation and ensured that data-flow dependencies were respected. However, ranks are natural numbers and give an over-constrained view of data-flow dependencies. In other terms, ranks are already a refinement of the minimal partial order consisting only of data-flow dependencies. While this approach was appropriate in the context of providing a total execution order of blocks, as it was the case for sequencer O, it was difficult to change the implementation to let it represent partial orders.

There are many optimized data-structures for dynamically computing transitive reachability [9]. We implemented a simple graph data-structure based on a dense matrix representation with a fixed set of vertices, for the incremental computation of both the transitive closure and transitive reduction of a graph while dynamically adding links.

Thus, the size required for a matrix with n nodes is $\Theta(n^2)$. Adding links between nodes requires a propagation step which costs $\Theta(n^2)$ in time. This propagation step also detects circular dependencies and maintains both the transitive closure and the transitive reduction of dependencies. Getting the list of all pairs of blocks that are unrelated has a complexity of $\Theta(n^2)$. Checking whether current order is total is a constant operation. In practice, each cell in the matrix requires 2 bits of memory[5]. The number of blocks in a subsystem is generally low enough to not be problematic with respect to the space complexity of our approach.

2.3.1 Invariants and properties

A graph of n nodes is represented by a square matrix M of size n^2. Each number k from 0 to $n-1$ is associated to a node named n_k. We note $<$ the strict order between nodes that is represented by a graph. Each cell $M_{i,j}$ of the matrix represents whether the relationship $n_i < n_j$ holds between nodes n_i and n_j. $M_{i,j}$ is one of the three following values: *none*, *direct* or *indirect*. We ensure that the following invariants hold in a matrix: (i) a value of *none* at cell $M_{x,z}$ means that $n_x < n_z$ does not hold; (ii) when the $n_x < n_z$ relationship holds, then $M_{x,z}$ is *direct* if and only if there is no indice y such that $n_x < n_y$ and $n_y < n_z$; (iii) otherwise, $M_{x,z}$ is *indirect*. Moreover, the graph is never allowed to contain circular dependencies. The implementation described hereafter maintains the above invariants, which give us the following properties: (1) The set of cells where $M_{i,j} \in \{\textit{direct}, \textit{indirect}\}$ represents the transitive closure of the $<$ dependency relationship. For all indices i and j, a non-*none* value at $M_{i,j}$ means that $n_i < n_j$. (2) The set of cells where $M_{i,j} = \textit{direct}$ is a transitive reduction of the $<$ dependency relationship. For all pair of indices (i, j) such that $M_{i,j}$ is *indirect*, there is a sequence of indices $k_1, ..., k_p$ $(p > 0)$ such as $M_{i,k_1} = M_{k_1,k_2} = \cdots = M_{k_p,j} = \textit{direct}$. (3) If and only if the order represented by $<$ is total, then for all indices i, j either $M_{i,j} = \textit{none}$ or $M_{j,i} = \textit{none}$, but not both. In other words, for a matrix of size n, then $<$ is total if and only if there are exactly $\frac{n(n-1)}{2}$ non-*none* values in the matrix (more than this number would imply a circular path; less would leave at least a pair of nodes unrelated). We rely on the third property above to easily check whether an order is total.

2.3.2 Link addition in a graph

A graph structure is implemented as an Ada package where size is a generic parameter. A connectivity matrix of size n^2 is initialized with *none* values. Also, a variable named *Remaining_Cells* is initialized to $\frac{n(n-1)}{2}$. Everytime a *none* cell in the matrix is changed into another value, *Remaining_Cells* is decreased. Thus, we can implement *Is_Total* as a function which returns whether *Remaining_Cells* is zero. Adding a link between two nodes requires a propagation mechanism to maintain the invariants previously seen. Link addition is split into two procedures, *Basic_Link*, which checks for any circular dependency and decrements *Remaining_Cells*, and *Link*, which propagates the relationship being added to all predecessors and successors. The *Check_Cycles* procedure raises an exception in case of circular dependencies and is defined as follows:

```
procedure Check_Cycles
(Self : in out Graph; From, To : Matrix_Index)
is
begin
   if From = To or Self.Matrix (To, From) > NONE then
      raise Cyclic_Graph;
   end if;
end Check_Cycles;
```

The above is a simplified version of the actual code, which also logs the offending cyclic path. *Basic_Link* is defined as follows:

```
procedure Basic_Link (Self : in out Graph;
                      From, To : Matrix_Index;
                      Link : Link_Type)
is
  Previous : constant Dependency :=
    Self.Matrix (From, To);
begin
   if Previous = NONE then
      Self.Check_Cycles (From, To);
      Self.Remaining_Cells := Self.Remaining_Cells - 1;
   end if;
   if Previous = DIRECT or Previous = NONE then
      Self.Matrix (From, To) := Link;
   end if;
end Basic_Link;
```

Link_Type is a subtype of *Dependency* with excludes *none*. The above procedure ensures that it is not possible to put a

[5] Actual size reported by GNAT when providing the *Pack* directive.

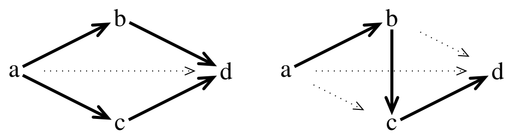
direct link if the link was previously known to be *indirect*. Indeed, we only add a *direct* link if there was previously no link between nodes, to be sure that cells with a *direct* value represent the minimal set of edges covering the whole graph. It is also possible to change a *direct* link to an *indirect* one, during the propagation initiated by the addition of a new, more *direct* link. The previously *direct* relationship can then be deduced transitively, which is why it should be replaced by an indirect one. This is done by propagating indirect relationships inside the *Link* procedure below (the *Self.Precedes* function simply checks that there is a non-*none* value in the matrix).

```
procedure Link
(Self : in out Graph; From : Matrix_Index;
 To : Matrix_Index; Link : Link_Type := DIRECT)
is
  Previous : constant Dependency :=
    Self.Matrix (From, To);
begin
 -- Add the "from < to" relationship
 Self.Basic_Link (From, To, Link);
 if Previous > NONE then
   return; -- Exit early (*)
 end if;

 --  Compute transitive connectivity
 for Y in Matrix_Index'Range loop
  for X in Matrix_Index'Range loop
   declare
    DX : constant Boolean := Self.Precedes (X, From);
    DY : constant Boolean := Self.Precedes (To, Y);
   begin
    if DX then
     -- (x < from) => (x < to)
     Self.Basic_Link (X, To, INDIRECT);
    end if;
    if DY then
     -- (to < y) => (from < y)
     Self.Basic_Link (From, Y, INDIRECT);
    end if;
    if DX and DY then
     -- (x < from),(to < y) => (x < y)
     Self.Basic_Link (X, Y, INDIRECT);
    end if;
   end;
  end loop;
 end loop;
end Link;
```

The above procedure assumes that the graph initially respects the properties expressed in the previous sections and ensures that they hold after adding the new link. When the previous value in our matrix at indices *From* and *To* was *none*, we can exit the procedure early *(*)*. Indeed, by construction a *none* value means that there are no direct or indirect relationship such as *From* $<$ *To*. Moreover, if previously the inverse relationship *To* $<$ *From* held, then *Basic_Link* would have raised an exception due to a cyclic dependency, which is not the case at this point. We can then safely assume that no link propagation is required.

The two nested loops add the required links between predecessors and successors of our nodes. Even though this can seem counter-intuitive, nodes previously known as *direct* are unconditionally changed into *indirect* ones. This is because the new *direct* link being added replaces a *none* value in the matrix, which, thanks to cycle detection, implies that the two nodes being linked were previously unrelated. Hence, the new link is currently the only one that allows to link *From* and *To* and cannot be removed. However, if a predecessor x of *From* used to be a *direct* predecessor of a successor y of *To*, then we have to change it to an *indirect* one since now there is another path from x to y



(a) Initial graph (b) Addition of $b < c$

Figure 2: *Direct* links (thick) being changed as *indirect* ones (dotted).

(the previous *direct* link between x and y is not the unique one anymore). This does not have an impact on links which already were *indirect*. See figure 2 for an example of this behavior.

2.3.3 Internal nodes

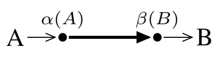
We allow to declare additional nodes in a graph, not bound to actual blocks. For example, an internal node could be used to model the set of all input ports of a block: the internal node would have outgoing links to each input port of the block and we could refer to this node whenever we want to add dependencies for all inputs. We use internal nodes to model trigger dependencies, as detailed in section 2.4.2. In order to handle those nodes, we allowed the size of the matrix n to be defined in terms of two parameters, b (blocks) and e (extra), such that $n = b + e$. The range $0..(b-1)$ represent nodes associated with actual blocks, called *block nodes*, whereas $b..(n-1)$ represents additional nodes called *internal nodes*: only block nodes are taken into account when checking whether current order is total. The overall square matrix is divided into two main zones: a square matrix B from $(0,0)$ to $(b-1,b-1)$ containing links between block nodes, and the remaining L-shaped region I containing links involving at least one internal node. We updated the graph structure with a normalization operation which projects all links inside I as links in the B area. This operation may change *indirect* links to *direct* ones in B and clears area I, while preserving the previous properties. Normalization is automatically applied when collecting all links, before saving current graph. It first clears the I area by resetting all cells to *none*, then recompute actual dependencies for block nodes. Computing the actual dependencies has an effect only on matrix cells $M_{x,z}$ that are set to *indirect*. If there is no index y such that $n_x < n_y < n_z$, then the dependency is changed as a *direct* one.

2.4 Converting dependencies as links

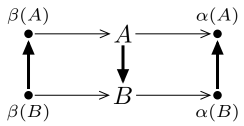
Thanks to the graph structure defined previously, the role of the sequencer is simplified: we only need to convert block relationships and attributes as links. We first show the general approach used when adding dependencies as graph, and then discuss the particular case of trigger dependencies.

2.4.1 Two-step refinement

We strictly apply the following two-step approach when refining a graph according to a sort criterion: *first*, compute the list of all pairs of blocks that are currently unrelated; *then* try to add a link between each pair of blocks according to current criterion. This separation allows us to prevent modifying the graph while looking for unrelated pairs of blocks, which would let the order by which pairs are visited influence block sequencing. For example, let A, B and C



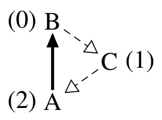
(a) Data signals



(b) Control signals

Figure 3: Converting data and control signals as dependencies

be three blocks; a data-flow signal connects A to B. With $u(x)$ being defined as the user-defined priority of a block x, we set block priorities as follows: $u(A) = 2$, $u(B) = 0$ and $u(C) = 1$.



According to priorities, we have both $C < A$ and $B < C$. In sequencer N, when introducing user-defined priorities, we first collect both (A, C) and (B, C) and then try to add links for each pair. This leads to a cyclic path in the graph which is reported to the user. With the original sequencer O, comparisons are made in an arbitrary order, which depends on implementation details, and eventually give either (A, B, C) or (C, A, B) as an execution order: instead of reporting a circular dependency, sequencing terminates normally. Note however that the order by which sort criteria are processed in a sort policy matters, because some of them are intended to refine a graph (e.g. priorities) whereas others unconditionally add all possible dependencies (e.g. data-flow dependencies).

2.4.2 Trigger dependencies

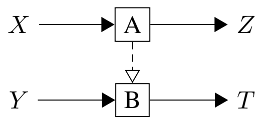
For each block b, we denote $dp(b)$ (resp. $cp(b)$) the set of direct predecessor blocks according to data-flow (resp. control-flow) dependencies. Those sets are determined by visiting incoming control-flow and data-flow signals, while ignoring blocks like *UnitDelay* which have no instantaneous dependency between input and output ports.

We convert data-flow and control-flow dependencies as links inside our dependency matrix. In order to do so, we need to propagate control-flow dependencies so that block activation follows a function-call semantics: when a block A is activated, any block B it controls is also activated and executed during the execution of block A. That means that (i) all the predecessor blocks in $dp(B)$ must already have been executed, and (ii) no block C such that $B \in dp(C)$ can be executed before A itself finishes its execution.

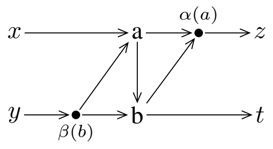
In order to express those constraints, we introduce intermediate nodes in our matrix. For all blocks, we define $\beta(B)$ ("before") as either the node B or the node which is directly preceding B with respect to node dependencies. Likewise $\alpha(B)$ ("after") is either B itself or the node directly succeeding B. Here, "directly" means that by construction we guarantee that no other node is ever scheduled between an intermediate node and the node of the block it represents.

By definition, $\beta(B) = B$ for all block B such that $cp(B) = \emptyset$, and $\alpha(A) = A$ for all block A such as there is no block B in current system block such as $A \in cp(B)$. In other cases, a new node is created and linked to its associated block.

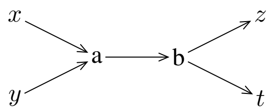
Once those nodes are created, we iterate over all blocks in current system and transform data and control signals as two different patterns of links, as shown in figure 3. Data-flow signals between blocks A and B simply add a link between $\alpha(A)$ and $\beta(B)$. Control-flow signals are translated in such a way that predecessor (resp. successor) blocks of controlled blocks are placed as predecessors (resp. successors) of the controlling blocks. In figure 3b we can see that a trigger between blocks A and B introduces links: (i) between A and B, because B cannot be computed before A, (ii) between $\beta(B)$ and $\beta(A)$ and (iii) between $\alpha(B)$ and $\alpha(A)$. This transformation scheme is done locally, for each block, but dependencies are eventually applied transitively by the underlying data-structure. For example, if block B was controlling a block C, the predecessors and successors of C would be scheduled respectively before and after A. Figure 4 shows an example of transformation from the original data-flow diagram to the resulting dependency graph.



(a) Original data-flow diagram with a control signal going from A and controlling the activation of B (dashed arrow).



(b) Dependency graph built according to transformation rules. $\forall n \in x, y, z, t$, $\beta(n) = \alpha(n) = n$. Also $\beta(a) = a$ and $\alpha(b) = b$.



(c) Equivalent data-flow graph after transitive reduction, as computed by the sequencer.

Figure 4: Elimination of trigger dependencies

3 Conversion of System Models

We provide a transformation mechanism from the System Model subset of P to SSME. For that purpose, we also developed a high-level library on top of the generated EMF classes.

3.1 Definitions

For each element p in the P language we define $\langle p\rangle$ the unique identifier of p. In practice, $\langle p\rangle$ is built from the unique XMI identifier of element p, which is always the string representation of a natural number, prefixed with "P". This naming scheme produces valid SSME identifiers which can be used to link Signal elements to the original P elements from which they are generated. Also, for any symbol m in $\{\mathit{label}, \mathit{local}\}$, $\langle p\rangle_m$ is a valid Signal identifier derived from $\langle p\rangle$ but distinct from it.

The translation of an element x of type T as an SSME element S, with a modifier m and with respect to an environment E, is defined by equations of the form $T_m(E, x{:}T) = S$. The result of applying the translation is denoted $T_m(E, x)$. Modifiers are a way to implement return-type polymorphism, where the behavior of T is specialized according to desired type of the value to be returned. Not all modifiers are meaningful for all types of inputs. A modifier m is an optional symbol: we define $T(E, P) = T_\emptyset(E, P)$, where $\emptyset$ is the empty modifier. The returned type of the translation when using the empty modifier is either a process expression or a signal expression, whichever is the most relevant for a given parameter: arithmetic expressions are translated as signal expressions, whereas data-flow connections are translated as equations.

An environment E is a set of zero or more bindings $s_i \mapsto v_i$ from symbols s_i to values v_i. We note $E.s$ the value bound to symbol s in environment E. We define $F = E\left[v_1 \mapsto s_1, \ldots, v_n \mapsto s_n\right]$ any environment F derived from E such that for all i between 1 and n, $F.s_i = v_i$; for any symbol w different from all symbols s_1 to s_n, $F.w = E.w$. For conciseness, $T_m(P)$ represents $T_m(E, P)$ when the environment E can be unambiguously inferred from the context. There is an implicit default environment from which other environments are derived where global options are defined.

3.2 General approach

The conversion function is recursively applied on P hierarchical data-flow graphs. Subsystems and blocks are translated as two nested Signal processes where input and output ports are represented by input and output signals. The outer process accepts inputs and outputs as given from the environment whereas the inner process is only called when a particular block is activated. The outer process is thus responsible for filtering inputs and providing default outputs according to the expected Simulink semantics of blocks. We define an auxiliary Signal process which dynamically computes if a block is *active* given the block's *control-port*, *enable* and *edge-enable* ports:

```
1  process simulink_control =
2  (? event tick, sample, cp; boolean en, eden;
3   ! event active)
4   (| ckeden := ^eden | cken := ^en
5    | enabled := (eden default tick) and
6                 (en default tick)
7    | active := cp ^+ ([:enabled]  ^* sample)
8    | active ^# ^0
9    | cp ^# (ckeden ^+ cken)
10   |) where event ckeden, cken; boolean enabled; end;
```

The above process also accepts input *tick* and *sample* events. The *sample* event is related to multirate models which are currently not considered in QGen: as a consequence, the *sample* event is always present in practice. The *simulink_control* process encodes some constraints related to blocks through clock relations: for example, line 9 represents the fact that if a block has a control-port, it cannot also have either an edge-enable or an enable port. Line 7 states that a block is active either if an event is present on its control-port or if the block is both *enabled* and at a tick where it should be computed. The *enabled* boolean is false soon as one of the enable (*en*) or edge-enable (*eden*) port has a false value (this value is computed elsewhere). The *[:enabled]* expression denotes an event that is present only when the boolean *enabled* is true. The above encoding can be used for all variations of system blocks, even if they do not have control-ports or enable ports. Indeed, when generating code for system blocks, it is sufficient to bind some input signals to the null clock to specify that a port is absent, which is then simplified by constant folding.

3.3 Dependencies and atomicity

We convert block-level dependencies as computed by QGen as signal dependencies between labelled process. A labelled process is a call of a process (e.g. a process representing a block) where all inputs and outputs are virtually associated with a label. The label, when used in a dependency, can be used to schedule all the input and output signals of a block before or after those of another process. We convert dependencies according to either the *nextExecutableBlocks* or *executionOrder* attributes of blocks. In order to model atomicity, we add a particular pragma, namely *Unexpanded*, to system blocks. This pragma is sufficient to instruct the Signal compiler to not interleave computations inside an atomic block with computations that exist outside that block.

3.4 System Model

A System Model element is a root element in P. Let s be a System Model containing n elements e_i, $0 \leq i < n$. s is translated in SSME as a list containing a single module M named $\langle s \rangle$ which contains a declaration D and and process definition P:

$$T(E, s : \textit{System Model}) = [M]$$

$$M = \textit{module}(\langle s \rangle\ , [D, P])$$

According to whether $E.type$ is *p* or *signal*, declaration D is respectively either (i) an import statement to a predefined Signal library, namely *import("P")* or (ii) a predefined list of external types τ: $\textit{type}(\tau, \textit{external})$. P is a process $q = \textit{process("main",S,B)}$ with a signature S and a process body B. The P process accepts as many input signals as the union of input ports of all e_i elements and provide as many outputs as the union of output ports. S is thus defined as $\textit{io}(\cup_{i=0}^{n} I_i, \cup_{i=0}^{n} O_i)$, where for each i such that $0 \leq i < n$:

$$I_i = \cup_{p \in \textit{In}(e_i)} T_{\textit{decl}}(p) \quad O_i = \cup_{q \in \textit{Out}(e_i)} T_{\textit{decl}}(q)$$

The body B is a new Signal process defined as:

$$B = \textit{restriction}(\textit{composition}([]), [])$$

We recall that Signal defines two kinds of expressions: (i) signal expressions, which are equations over data-flow variables and (ii) process expressions, which include notably composition and restriction processes. The B process is a *restriction* process, which holds both a sub-expression e and a set of lexically scoped declarations d. Declarations made in d are only visible in d and e. Here, e is a *composition* process, i.e. a set of zero or more parallel processes. The declarations and expressions in B are initially empty but eventually populated by the translation of inner elements. For all elements e_i contained in s, we apply $T(E\left[module \mapsto M, subproc \mapsto q\right], e_i)$. The *module* and *subproc* symbols respectively represent the root element of the generated SSME element and the process associated with current subssytem: here the "main" process is referenced.

3.5 System blocks

System blocks contain zero or more children blocks. A system block s converted as an expression represents a labelled call to the process representing s. The arguments of the process call are translations of the input ports of s, which are variables declared in current scope. Likewise, the mutliple results of the process call are bound to the signals resulting from the translations of output ports of s.

$$T(s\text{:}System) = labelled(\langle s\rangle_{label}, call(T_{decl}(s), I, O))$$

$$I = \cup_{p\in In(s)} T(p) \quad O = \cup_{q\in Out(s)} T(q)$$

The declaration associated with a system block is a process definition. Each system block is translated as one *control* process c calling a *body* process b. The control process c is responsible for determining if the block is currently active and feeds the internal body b with filtered inputs. Moreover, the outputs from b are repeated (or replaced by default values) so that c can provide outputs at the same rate as inputs are given: c takes care of merging outputs from b either with a default value or the previous value of each output port, according to each port's *resetWhenDisabled* attribute.

$$\begin{aligned}&T_{decl}(s\text{:}System) = c = node(\langle s\rangle, S, X)\\&X = restriction(Y, [D, ldecl])\\&Y = composition([ldeps, lclock, event, in, out, call])\end{aligned}$$

The signature S is computed as above by converting all inputs and outputs of s. The set of declarations of the control process c holds a label declaration *ldecl* and a declaration D for the body process b, detailed thereafter. There are 6 parallel processes being composed, each of them having a specific purpose: *call* is a process call to the body process b; *ldeps* holds a list of inter-label dependencies, which allows to encode the partial order provided by the input model; *lclock* contains clock equalities for labels, indicating that all contained blocks are executed synchronously; *event* calls the *simulink_control* external process and contained translations of control, enable and edge-enable ports according to their attributes; *in* and *out* correspond respectively to filtering and merging equations, where local variables are declared for all input and output data-ports. That process b is declared in D in a modified environment F:

$$F = E\left[\begin{array}{ll} subproc \mapsto b, & ll \mapsto ldecl,\\ ls \mapsto ldeps, & lc \mapsto lclock\end{array}\right]$$

In addition to *subproc*, other symbols keep a reference to other parts of the control process where *convert* can add information for each block contained in s. Then, D holds a process definition for b with the same input/output signature as c and the recursive conversion of each block of s as expressions in environment F. During this conversion, *convert* populates the labels declarations and constraints held in the control process c.

3.6 Elementary blocks

By default, elementary blocks are expressed as external processes in Signal. The reason for this is that we do not aim to provide a complete implementation of Simulink's semantics in parallel to the one currently implemented by QGen. Indeed, there are over a hundred of blocks defined in QGen's default Block library and each of them allows multiple set of configurations that may have an important impact on their behavior. For example, the *Sum* block can process scalars, vectors and matrices depending on the data linked to its input ports. We would rather reuse the existing code generation mechanism of the compiler to produce target code. Indeed, QGen's Block library defines, for each block, a set of functions that provide the imperative statements implementing the different steps of computation of this block, namely *Make_Memory_Variables()*, *Get_InitStatements()*, *Get_ComputeStatements()* and *Get_MemUpdateStatements()*. Those functions encode the actual semantics of each block according to its parameters and provide statements or declarations in a subset of P expressing code, called Code Model (e.g. statements, operators, functions). Thus, we are interested in converting the generated Code Model elements as signal expressions instead of providing an ad-hoc implementation of each possible block. We currently support only a small subset of Code Model elements. The approach consisting in compiling Code Model elements requires to build a control-flow graph, as commonly done in compilers [16]. However, we cannot currently assume that Code Model programs generated by QGen are in a static single assignment (SSA) form [7], which would simplify data-flow analysis [18] and could let us exploit our previous results with translation from C/C++ to Signal [2].

3.7 Datatypes

We propose two different ways of translating P data-types to SSME. First, we can treat all types as external types and translate mathematical operations as call to external processes. Alternatively, we can translate types as Signal types whenever possible, which tend to produce simpler code that does not depend on the external library of types provided by QGen. In both cases, it is sometimes necessary to convert values directly as Signal constants, like in array indices for which QGen does not define a specialized type. There are however limitations with this approach, because Signal does not define unsigned types nor 64 bits integers. The translation is straightforward, except that when an unsigned type is requested, we use a larger signed type so that all values can be represented. Also, we produce warnings if the required number of bits is too large for Signal.

4 Validation and applications

We validated our approach with the test suite used by QGen which is composed of over two-hundred small-sized Simulink models. We tested both block sequencing and model transformations.

4.1 Side-by-side testing of sequencers

We ensured that our implementation did not regress from the previous one by (i) developing a dedicated sorting policy and a ranking function for our implementation that replicated the behavior of the existing sequencer, and (ii) comparing the results of both implementations with side-by-side tests. Those tests showed discrepancies for exactly ten models which were acknowledged to be due to some defects in sequencer O: (i) the failure to reject models where some blocks cannot be unambiguously sorted, either because there were circular dependencies or because blocks were treated as equal (we found out that the tie-breaking comparison functions on block names could fail because systems could contain blocks having the same name after preprocessing; this was fixed by using the fully qualified name of blocks) and (ii) the incompatibility between the computed order and the compilation strategy regarding triggered blocks (sequencer O would produce an order which would not describe how triggers are implemented).

4.2 From P to SSME

We managed to load and process a public use case provided by Rockwell Collins France composed of safety-critical components implementing display logic for helicopter data. The XMI file resulting from QGen's compilation weights 9.7MiB and is loaded as an EMF runtime object in our Clojure environment in over 9 seconds; Comparatively, translation to SSME takes between 1 and 2 seconds, which is the same time required for pretty-printing the same model as a Signal textual file; exporting the runtime model as an XMI document takes a little less than 500ms. The produced model is made of 287 Signal processes which mirror the hierarchical organization of the original model. It represents the data-flow and control-flow constraints among blocks inside the system. We can process that model with Polychrony in order to compute a flattened network of blocks grouped by common dependencies into 109 separate clusters of sequential code. Polychrony can produce multithreaded code from clusters where threads synchronize themselves either with a message-passing protocol or with signal/wait operations [3].

We also experimented with the balance drive controller model present in QGen's test suite. We modified the input model so that each subsystem is declared to be atomic in order to obtain a hierarchy of Signal processes after translation. Then, we collected all original P subsystems and generated a map from their unique identifiers to a unique natural number, starting from zero and increasing. With this temporary map, we dynamically added *RunOn* directives [3] in our SSME model to each of its processes in order to bind subsystems to distinct execution units. This manual step requires fifteen lines of interactive code and simulates a partitioning of our system according to a possible architectural description of it. Then, the distributed Signal model can be used to generate distributed code that complies with *RunOn* directives. An automatic translation of architectural P elements could be eventually feasible in future versions: even though there is an existing architecture description language in P which allows to declare processors and buses, as well as non-functional properties such as the period and deadline of a task, there is currently no way to map System Models to architectural ones.

4.3 From SSME to P

Our original intent when integrating Signal and QGen was to use Signal as a model optimizer for P, with respect to static block scheduling, timing and architectural properties. For example, starting from a multirate model, i.e. a model where blocks are sampled at different periods, we could group blocks into different asynchronous components while preserving or adding synchronization flows when necessary. In order to perform this task, our tool is expected to return modified P models. In practice, we effectively have access to a model m, its translation s in SSME and the model s_p obtained by running Polychrony on s with specific parameters p. With the known mapping between m and s, we can determine whether an element in s_p was originally present in s or if that element is introduced by Polychrony itself. However, we do not have a systematic transformation scheme to build a meaningful copy of m taking into account the modifications of s brought by s_p at the model level. Instead of trying to modify existing models, we are now implementing a general purpose transformation from Signal to P, as part of Polychrony, which systematically converts endochronous processes as a hierarchy of triggered subsystems following the original clock hierarchy. Alternatively, it would be easier to directly generate P Code Model instead of System Models.

5 Conclusion

We developed an alternative block sequencer for QGen for the purpose of computing both partial and total orders from input models. The purpose of this sequencer is to allow QGen to interoperate with external sequencing tools while providing guarantees about the compatibility of external block execution orders with respect to both QGen's compilation scheme and user expectations. This sequencer is available as a separate tool and not fully integrated inside QGen. Our work contributed nonetheless to test and fix the existing codebase of QGen.

We also presented a model transformation tool from the P language used inside the QGen model compiler to the SSME language representing synchronous Signal programs. This work is based on a high-level API designed on top of SSME and can be used to transform a subset of Simulink to Signal. We ran the conversion tool and the set of models used by QGen for its regression tests and successfully converted medium to large models. The P language is capable of representing a useful subset of Simulink. That is why it is an interesting tool to help interpreting Simulink models and possibly architectural properties as executable Signal programs. Our perspective for this tool is to add support for the conversion of more Code Model elements, as generated by QGen, in order to produce executable programs with Signal. The programs currently produced with our transformation tool can be compiled by Polychrony and reorganized as clusters of smaller processes. We expect QGen to eventually provide an architecture description language inside P, and our perspective with regard to P remains to be able to automatically distribute models given some of their architectural properties while preserving synchronization constraints.

6 Acknowledgments

We thank the anonymous reviewers and especially our "shepherd" Eric Jenn for their helpful feedback.

References

[1] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events

and relations: the SIGNAL language and its semantics. *Science of computer programming*, 16(2):103–149, 1991.

[2] Loï Besnard, Thierry Gautier, Matthieu Moy, Jean-Pierre Talpin, Kenneth Johnson, and Florence Maraninchi. Automatic translation of C/C++ parallel code into synchronous formalism using an SSA intermediate form. In *Ninth International Workshop on Automated Verification of Cirtical Systems (AVOCS'09)*, 2009.

[3] Loïc Besnard, Thierry Gautier, Paul Le Guernic, and Jean-Pierre Talpin. Compilation of polychronous data flow equations. In *Synthesis of Embedded Software*, pages 1–40. Springer, 2010.

[4] Matteo Bordin and Franco Gasperoni. Towards verifying model compilers. In *5th International Congress and exhibition ERTS2*, 2010.

[5] Matteo Bordin, Tonu Naks, Andres Toom, and Marc Pantel. Compilation of heterogeneous models: Motivations and challenges. In *European symposium on Real Time Software and Systems (ERTS), Toulouse*, volume 29, pages 08–01. Citeseer, 2008.

[6] Alan Burns. The ravenscar profile. *ACM SIGAda Ada Letters*, 19(4):49–52, 1999.

[7] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13(4):451–490, October 1991.

[8] James B Dabney and Thomas L Harman. *Mastering simulink*. Pearson, 2004.

[9] Camil Demetrescu and Giuseppe F. Italiano. Dynamic shortest paths and transitive closure: Algorithmic techniques and data structures. *Journal of Discrete Algorithms*, 4(3):353 – 383, 2006. Special issue in honour of Giorgio Ausiello.

[10] Francois-Xavier Dormoy. Scade 6: a model based solution for safety critical software development. In *Proceedings of the 4th European Congress on Embedded Real Time Software (ERTS'08)*, pages 1–9, 2008.

[11] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (AADL): An introduction. Technical report, DTIC Document, 2006.

[12] Oussama Feki, Thierry Grandpierre, Mohamed Akil, Nouri Masmoudi, and Yves Sorel. SynDEx-Mix: A hardware/software partitioning CAD tool. In *Sciences and Techniques of Automatic Control and Computer Engineering (STA), 15th International Conference*, pages 247–252. IEEE, 2014.

[13] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[14] Nassima Izerrouken, Olivier Ssi Yan Kai, Marc Pantel, and Xavier Thirioux. Use of formal methods for building qualified code generator for safer automotive systems. In *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness & Safety*, pages 53–56. ACM, 2010.

[15] Nassima Izerrouken, Xavier Thirioux, Marc Pantel, and Martin Strecker. Certifying an Automated Code Generator Using Formal Tools : Preliminary Experiments in the GeneAuto Project. In *European Congress on Embedded Real-Time Software (ERTS), Toulouse, 29/01/2008-01/02/2008*, 2008.

[16] Uday Khedker, Amitabha Sanyal, and Bageshri Sathe. *Data flow analysis: theory and practice*. CRC Press, 2009.

[17] Christophe Lavarenne, Omar Seghrouchni, Yves Sorel, and Michel Sorine. The syndex software environment for real-time distributed systems design and implementation. In *European Control Conference*, volume 2, pages 1684–1689. Citeseer, 1991.

[18] Alexandre Lenart, Christopher Sadler, and Sandeep K. S. Gupta. SSA-based flow-sensitive type analysis: combining constant and type propagation. In *Proceedings of the ACM Symposium on Applied Computing*, pages 813–817, 2000.

[19] Yue Ma, Jean-Pierre Talpin, and Thierry Gautier. Virtual prototyping AADL architectures in a polychronous model of computation. In *Formal Methods and Models for Co-Design, 2008. MEMOCODE 2008. 6th ACM/IEEE International Conference on*, pages 139–148. IEEE, 2008.

[20] MISRA-C MISRA. Guidelines for the use of the C Language in vehicle based software. *Motor Industry Research Association, UK*, 1998.

[21] Jorgiano Vidal, Florent De Lamotte, Guy Gogniat, Philippe Soulard, and Jean-Philippe Diguet. A co-design approach for embedded system modeling and code generation with UML and MARTE. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 226–231. IEEE, 2009.

Session 8

Design Space Exploration 1

Wednesday 27th, 14:45 – Ariane 2

Lean Model-Driven Development through Model-Interpretation: the CPAL design flow

Nicolas Navet, University of Luxembourg
Loïc Fejoz, RealTime-at-Work, France
Lionel Havet, RealTime-at-Work, France
Sebastian Altmeyer, University of Luxembourg

Abstract: We introduce a novel Model-Driven Development (MDD) flow which aims at more simplicity, more intuitive programming, quicker turnaround time and real-time predictability by leveraging the use of model-interpretation and providing the language abstractions needed to argue about the timing correctness on a high-level. The MDD flow is built around a language called Cyber-Physical Action Language (CPAL). CPAL serves to describe both the functional behaviour of activities (i.e., the code of the function itself) as well as the functional architecture of the system (i.e., the set of functions, how they are activated, and the data flows among the functions). CPAL is meant to support two use-cases. Firstly, CPAL is a *development and design space exploration* environment for CPS with main features being the formal description, the editing, graphical representation and simulation of CPS models. Secondly, CPAL is a real-time execution platform. The vision behind CPAL is that a model is executed and verified in simulation mode on a workstation and the same model can be later run on an embedded board with a timing-equivalent run-time time behaviour.

Keywords: Model-based design, programming language, model interpretation, design space exploration, simulation, timing predictability.

1 Model Driven Development with model as the code

Existing commercial MBD flows such as Matlab/Simulink® and Scade® successfully capture most of the aspects of model based design: requirements traceability, model design, simulation, code generation, test-cases generation, etc. Even though they are very powerful and successfully used in a wide range of industrial applications, these design flows do not cover all existing needs, be it only because they are complex and expensive.

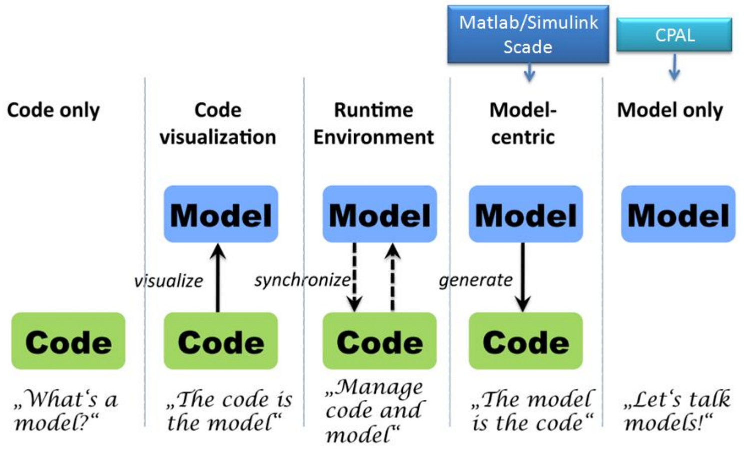


Figure 1: Spectrum of model-based design approaches (core of the figure from [Br04] and [Tr09]).

CPAL has been initially inspired by the success of three interpretation-based runtime environments, successfully certified at the highest criticality levels and deployed at large scale in railway interlocking systems over the last 20 years at SNCF (see [An12]) and RATP in France, and in UK and other countries through the Westlock interlocking system from Westingshouse. These technologies have proven to be technically successful in the sense that 100s of millions of people rely on them on a daily basis. They are however undisclosed proprietary technologies, specific to interlocking systems and have not been designed to meet the needs of most of today's and tomorrow's applications (Cyber-Physical Systems at large) and execution platforms (SOC, multicore, manycore).

Except for these applications and some implementations in industrial automata (PLCs), surprisingly, *model interpretation* has to the best of our knowledge not been widely explored yet for developing critical systems (see [Hu03] for one of the few notable works in that direction), albeit it possesses a number of key advantages:

- **Simplicity** for the end-user and quicker turnaround time: once designed and simulated the model can be uploaded to the target (e.g., by drag & drop).
- **Verifiability**: there is no discrepancy between the model and the code, and there are no software layers causing deviations from the expected temporal behavior at run-time.
- **Less error prone software**: because the total software size is greatly reduced both off-line and on the target (no operating systems, no compiler, no linker, etc). In particular, the interpreter is a thin software layer of a few thousand lines that can be more easily tested and verified than software made up of hundreds of thousands lines of code or more. In addition, as argued in [An12], the logic of the application is easier to verify since it is fully decoupled from the runtime services and written in a high-level language.
- **Cost-efficiency**: made possible thanks to the simplicity of the design flow and run-time environment.
- **Hardware-independence**: thanks to the ability of the interpretation layer to hide the complexity of the hardware platform from the programmer. A higher level of abstraction is important in light of the ongoing trend towards multi/manycore platforms and more heterogeneous execution platforms (SOC).

CPAL supports two types of model interpretation: the direct interpretation of the design models on an interpretation engine running on top of the hardware, called “bare-machine model interpretation” (BMMI), and the interpretation on top of an OS. The latter is less predictable from a timing point of view but more convenient for development and experimentations. In addition, the interpreter can re-use the interfaces to the I/O provided by the OS. Whatever the type of interpretation, there is a slowdown due to model interpretation. However, we believe that for a significant share of embedded systems, the simplified and accelerated model development (reduced time-to-market) will outweigh the overhead due to model interpretation on the target architecture. In addition, dedicated hardware support in FPGA or ASIC may offset part of the performance loss.

CPAL and associated tools are jointly developed by our research group at the University of Luxembourg and the company RTaW since 2011. The CPAL documentation, graphical editor and execution engine for Windows, Linux, embedded Linux and RaspberryPI are freely available for all uses at http://www.designcps.com. A BMMI port of CPAL is available for Freescale FRDM-K64F boards. A commercial version for embedded targets will be introduced progressively.

2 CPAL: providing high-level abstractions for embedded systems

The main requirement when designing CPAL was to natively provide the high-level abstractions familiar in the domain of embedded systems needed to express in an unambiguous and concise manner domain specific patterns of functional behaviors as well as non-functional properties. The concept of *process* is the core language entity to implement a recurrent activity having its own dynamic. A process is automatically activated at a specified rate, with the optional requirement that a specific logical condition is fulfilled to execute (this is called *guarded executions*). CPAL processes are classically referred to as tasks, runnables or threads in other contexts.

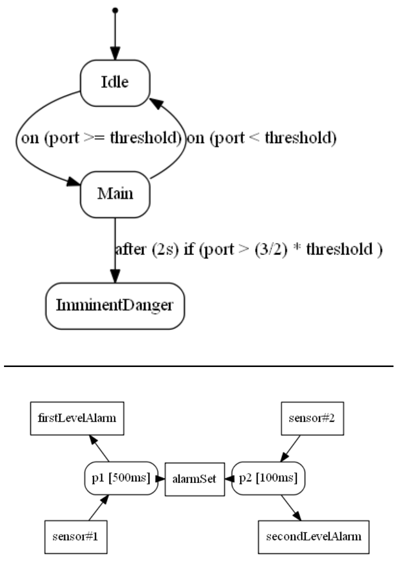
CPAL provides the programmers with high-level abstractions well suited for the domain of CPS such as

- *Real-time scheduling mechanisms*: processes are activated with a user-defined period, possibly with offset relationship with each other and additional execution conditions such as the occurrence of some external events.
- *Finite State Machines* (FSM): the logic of a process can be defined as a Finite State Machine (FSM) based on Mode-Automata [Ma03] where code can be executed in the states, or upon the firing of transitions. The semantics that is implemented in CPAL is to first execute a transition if possible and then execute the current state’s code which enables the control program to react faster on external events,
- *Communication channels* to support data flow exchanges between processes, and reading/writing to hardware ports. The semantics attached to a channel can be chosen to be FIFO or LIFO buffering, or data overwriting,
- *Introspection mechanisms* that enable processes to query at run-time their execution characteristics such as their activation rate and activation jitters. This feature is typically used to implement

control algorithms that must adapt to their frequency of execution or their execution jitters by compensating for them.

The abstract and concrete syntax of CPAL has been inspired by a number of diverse languages such as Eiffel, MISRA C and Erlang, model-based design products such as Matlab/Simulink® and Scade®, verification frameworks such as Promela/Spin and more generally what is usually referred to as the synchronous programming approach, such as Giotto [He03]. However, CPAL has been designed with the requirement to remain a small, simple and unambiguous language, easy to start with for the C or Java programmer, and less demanding than the synchronous programming models.

The example CPAL program below defines a monitoring process which, when a threshold on the measured quantity is exceeded, signals an abnormal behavior and, after a certain time above another threshold, sets an alarm. When this happens, another process starts then being executed at a higher rate to confirm with measurements from another sensor the alarm condition. This can then be used by a supervision process to take the appropriate measures (e.g., error recovery or error mitigation).



```
processdef MonitorProc(in uint8: port, out bool: abnormal, out bool: danger)
{
  const uint8: threshold = 30;
  state Idle{ /* ... */ }
  on (port >= threshold) to Main;

  state Main {
    abnormal = (port > threshold);
    /* ... */
  }
  after (2s) if (port > (3/2) * threshold ) {
    danger = true;
  } to ImminentDanger;
  on (port < threshold) to Idle;

  state ImminentDanger {/* ... */}
}

var uint8: sensor#1; /* Mapped to some I/O port and updated */
var uint8: sensor#2; /* upon activation of the processes */
var bool: alarmSet = false;
var bool: firstLevelAlarm = false;
var bool: secondLevelAlarm = false;

/* Instantiation of periodic monitoring processes*/
process MonitorProc: p1[500ms](sensor#1, alarmSet, firstLevelAlarm);
/* Guarded execution: second process is only executed when firstLevelAlarm is true */
process MonitorProc: p2[100ms][firstLevelAlarm](sensor#2, alarmSet, secondLevelAlarm);
```

Figure 3: Example CPAL program illustrating the concepts of input and output ports, native support for FSM, conditional and timed transitions and periodic process activation (with and without guard). The top-left graphic is the representation of the FSM embedded in the monitoring process, while the bottom-left graphic is the functional architecture with the flows of data, as both seen in the CPAL-Editor.

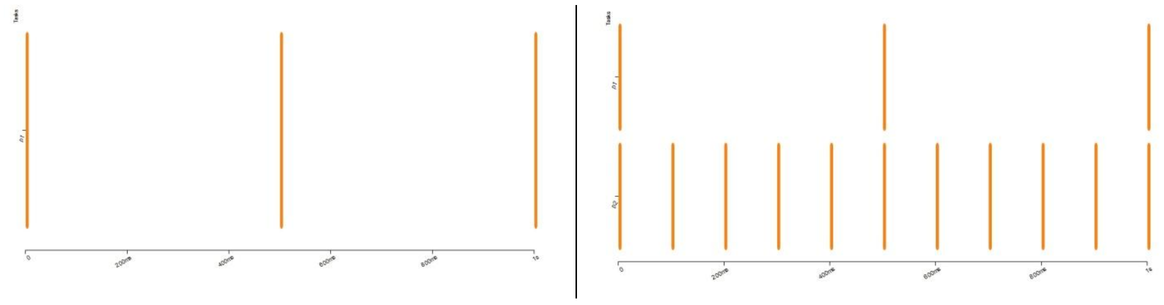


Figure 4: Gantt diagram of the activation of the processes as seen during execution. On the left, a single process is executed while, on the left, the second and more frequent process is being executed too because an alarm condition was signaled by the first one (screenshots from the CPAL-Editor).

3 CPAL to model, validate and execute embedded systems

3.1 CPAL use-cases

CPAL supports several use-cases discussed below.

3.1.1 High-level programming language for network simulation environments

CPAL can serve to describe the functional behavior of applications and high-level protocol layers. A CPAL model is for instance used in [Se15] to simulate the SOME/IP Service Discovery protocol in a Daimler Car's prototype network. Another CPAL program, available in [Fe15], implements the transmission of a video stream with segmented messages on an Ethernet network. The model hands over the frames once created to the simulation kernel of RTaW-Pegase, a communication architecture performance analysis tool from RTaW. Interestingly, the same CPAL simulation model can be executed with no changes on an embedded target or a workstation to experiment on a testbed later in the design process.

3.1.2 Modeling and simulation language for Design Space Exploration

CPAL is meant to support the formal description, the editing, graphical representation and simulation of cyber-physical systems. It can be used in its own development environment, like done for the FMTV Challenge [Al15], or within Matlab/Simulink to implement the controller, as done for the landing gear case-study [Bo14,Na14]. The simulation models can be executed in real-time (i.e., activation periods are respected) or as fast as possible in simulation mode. Simulation mode CPAL interpreters are available on Windows and Linux. This case-study is illustrated on the development of a smart parachute for UAVs in [Ci16] and further discussed in §3.2.

3.1.3 Real-time execution engine and overheads data

The intention of CPAL is to provide not only a modeling language, but also an interpreter which ensures equivalence between the simulated behavior of the model and the behavior on the execution platform. There are CPAL interpreters in real-time mode available for embedded Linux, Raspberry Pi and a BMMI port for the Freescale FRDM-K64F board which is based on a CortexM4 processor. On this latter inexpensive platform at 120Mhz (floating point enabled), the overheads we measured with a logic analyzer, or we calculated based on the code, are the following:

- the maximum activation jitter for periodic activation is 40us,
- the timer interrupts which occur periodically during the execution of processes takes less than 70 cycles, that is less than 0.6us,
- the time to decide the next process to execute and create future instances is 200 cycles + n * 560 cycles, that is 1.6us + n * 4.6us, where n is the number of process instances currently active,
- in-between process overhead is 2us maximum.

CPAL models are interpreted at run-time which involves a significant performance loss with respect to compiled code, typically a slowdown factor larger than 3. For CPS requiring maximal performances, code generation from CPAL or hooks to call native object code from CPAL processes would be feasible options. This latter technique seems promising to us since it enables to keep most of the additional control and monitoring capabilities of interpretation while allowing the re-use of legacy code and a close to compiled-code execution speed.

3.1.4 CPAL for learning and teaching

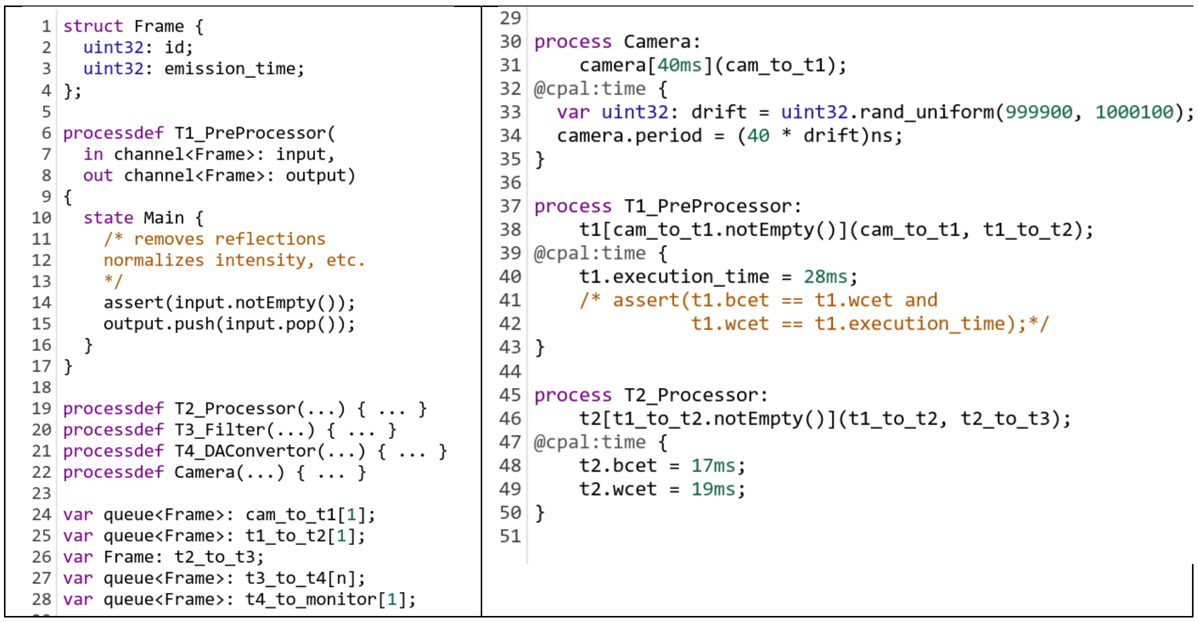
CPAL has been used for teaching since 2012 at our University at the 3rd year Bachelor level. CPAL is used to teach model-based design (MBD) for embedded systems with practicals such as programming a capsule coffee machine, a simplified programmable floor robot and elevator control system, etc. Our experience has been positive in terms of how fast students have been able to work autonomously on the development of the system. Indeed, most students are to master the language within a few hours. In addition to the simplicity of the language, the free availability of the tools, the on-line examples and the CPAL-Playground facilitate the learning process. Improvements ahead of us include a better tool support for 1) methodological processes such the ability to link design artifacts with requirements and 2) verification tools that exist at a prototypical stage (WCET and response time analyzers, state-space exploration).

3.2 Solving the FMTV challenge 2015

The Formal Methods for Timing Verification (FMTV) Challenge 2015 is a schedulability analysis problem proposed by Thales that challenges current techniques and tools from the timing verification community. The challenge is built around an aerial video tracking system and consists of 4 sub-challenges. A number of solutions using different formalisms were presented at the WATERS Workshop 2015 (see [Wa15]). To solve the sub-challenges, we proposed solutions relying on CPAL for the modeling and the simulation along with manual schedulability analysis. The reader is referred to [Al15] for a more comprehensive description.

Our key takeaways from the FMTV2015 challenge are the following:

- The modeling efforts were limited and the complete models were written in CPAL within less than 3 hours, which was significantly faster than the development of the automata based models.
- The CPAL model, along with the associated graphical representations, reveals ambiguities in the description and thus forces the system designer to consider each important aspect of the modeled system.
- The simulation capacity of CPAL allows to explore the timing behavior at design-time and to validate or disprove assumptions about it. For instance, deriving the solution for the first challenge by hand proved to be error-prone, and the use of simulation was helpful to better understand the dynamics of the system, and specifically check whether the worst-case conditions we devised could actually happen.
- If worst-case behaviors are looked for by simulation, simulation should be biased to explore parts of the search space we know are likely to contain such behaviors. For instance, in order to increase the likelihood to meet unfavorable scheduling scenarios, we used a random number generator that gave higher probability to the bounds of the interval, instead of a uniform distribution. This simple strategy was effective in creating situations leading to the maximum interferences in our experiments on the first challenge. This is however clearly an open research problem.
- With the help of a simple utility it is possible to extract from the CPAL model the characteristics of the tasks and automate the schedulability analysis. However, we were unable and do not see how to answer in an automated manner complex questions like asked in sub-challenges 1A and 1B without resorting to ad-hoc analyses. Identifying the scope of what can be fully, or partially, automated is in our view a question that deserves future work.



```
struct Frame {
  uint32: id;
  uint32: emission_time;
};

processdef T1_PreProcessor(
  in channel<Frame>: input,
  out channel<Frame>: output)
{
  state Main {
    /* removes reflections
    normalizes intensity, etc.
    */
    assert(input.notEmpty());
    output.push(input.pop());
  }
}

processdef T2_Processor(...) { ... }
processdef T3_Filter(...) { ... }
processdef T4_DAConvertor(...) { ... }
processdef Camera(...) { ... }

var queue<Frame>: cam_to_t1[1];
var queue<Frame>: t1_to_t2[1];
var Frame: t2_to_t3;
var queue<Frame>: t3_to_t4[n];
var queue<Frame>: t4_to_monitor[1];

process Camera:
    camera[40ms](cam_to_t1);
@cpal:time {
  var uint32: drift = uint32.rand_uniform(999900, 1000100);
  camera.period = (40 * drift)ns;
}

process T1_PreProcessor:
    t1[cam_to_t1.notEmpty()](cam_to_t1, t1_to_t2);
@cpal:time {
    t1.execution_time = 28ms;
    /* assert(t1.bcet == t1.wcet and
              t1.wcet == t1.execution_time);*/
}

process T2_Processor:
    t2[t1_to_t2.notEmpty()](t1_to_t2, t2_to_t3);
@cpal:time {
    t2.bcet = 17ms;
    t2.wcet = 19ms;
}
```

Figure 5: Excerpt of the CPAL Code for Thales FMTV Challenge 1. Process activation conditions are specified at the definition of the processes (e.g., *t1* is activated upon the arrival of a frame from the camera). The annotations in the comments are used for the simulation and the analysis of the model. The complete code is available at http://www.designcps.com/wp-content/uploads/fmtv15.zip.

Verification outside schedulability analysis (see §4.3) in CPAL currently relies on simulation. The complete language is not amenable to formal verification by model-checking or theorem proving. Although simulation does not offer exhaustive evaluation and hence, does not equate to formal verification, it is applicable on systems of any size. Simulation of CPAL code is timing accurate through the use of timing annotations (see line 40 in Figure 5 for instance), which can be derived by measurements on target architecture or WCET analyses. We believe also that heuristics, such as biasing random number generation towards the bounds of the interval as we experimented in [Al15], and search-intensive algorithms (see [DA14]) are promising techniques to efficiently direct the simulation towards unfavorable trajectories of the system. Although this remains to be demonstrated, such worst-case oriented simulation may be a practical alternative to formal verification for some systems, especially early in the design phases.

4 Scheduling and timing correctness

4.1 Timing predictability in CPAL

The correctness of a cyber-physical system usually does not only depend on its functional behavior, but also on its timing behavior. Similarly to functional determinism, i.e., the same input always leads to the same output, we may want systems where events occurs at pre-determined points in time. This notion of *time determinism*, at the heart of the synchronous approaches, is for instance discussed and advocated in [He08].

Modern architectures with history-sensitive components such as caches and buffers, however, lead to significant variations of execution times and are increasingly complex to analyze. Despite the determinism of all individual hardware components, the complex interplay thereof appears non-deterministic if it cannot be fully comprehended. In addition, changing environmental conditions, such as temperature or EMI, will affect the functioning of the system. For instance, significant clock drifts are caused by varying temperatures [Mo11]. Complete time-deterministic systems as defined in [He08] are thus hard to achieve.

With respect to the synchronous programming models, CPAL implements a weaker version of time-determinism, still providing a form of timing-predictability sufficient in many applications while remaining closer to mainstay software development practices. Our experience is indeed that timing-correctness most often does not necessitate time-determinism. For most systems, it is sufficient if the timing of events respects a set of constraints specific to the needs of the cyber-physical system, thus allowing a substantial degree of freedom. For instance, a system may has to react to an input within a given time bound, the order of some events may be essential, or a computation may has to be repeated periodically with limited jitter. Several, distinct systems can exhibit distinct timing behaviors, which are all considered correct, and furthermore, systems can show substantial timing variations at run-time and still be considered correct. In any case, a time-deterministic system is not a necessity for timing correctness for all systems.

Instead of a fully time-deterministic system, the execution framework enforces a fixed and deterministic event ordering irrespective of the execution platform. The exact timing of an event may be subject to variations that can be evaluated by a schedulability analysis, but the order in which observable events, such as process invocation or process termination, happen shall be statically defined. We refer to this property as *event-order determinism*. This allows the CPAL program to be developed in simulation mode on a workstation and to be later run on an embedded board with an equally acceptable timing behavior. More fine-grained timing constraints such as deadline constraints can be verified with the help of schedulability analysis (see $4.3).

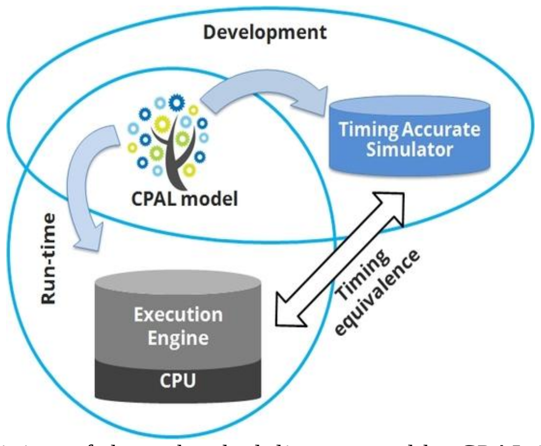


Figure 5: Event-order determinism of the task scheduling ensured by CPAL is a main dimension of timing equivalence between design time and run-time. Optionally, it is also possible to annotate a CPAL model with execution time information, obtained by measurements on target or WCET analysis, so as to achieve timing-accurate simulation of the model.

4.2 Scheduling model

If we solely concentrate on implementing a system's timing correctness, we can usually select from several scheduling policies and execution models. Among the various scheduling algorithm, we selected FIFO scheduling which is a predictable and lightweight policy particularly suited to our needs. Indeed, FIFO schedules processes non-preemptively and ensures *event-order determinism*, i.e., the order of process executions is defined statically and immutable. With this choice, we favor predictability of the run-time and simplicity of the execution engine over an optimized use of the computational resources.

In FIFO scheduling, processes are released strictly period and are executed in order of process release. Processes can be assigned priorities that serve as tie breakers in case of simultaneous process releases. Nevertheless FIFO scheduling is – in stark contrast to static-cyclic scheduling – a work-conserving scheduling policy which means that no CPU time is wasted. A system-wide clock is required to trigger process activation and to ensure determinism. All process release times are thus subject to the very same clock drifts, enforcing the unique execution order, but also restricting the system to uni-core processors or partioned multicore scheduling.

FIFO scheduling is well known to perform worse regarding schedulability than priority-driven dynamic scheduling policies such as rate-monotonic or earliest deadline first. For purely periodic systems like in CPAL, execution offsets [Mo12] can however be chosen so as to distribute the workload evenly over time, which significantly improves the ability of FIFO to meet real-time constraints [Alt15b]. Although FIFO fits well CPAL, the scheduling model of CPAL is not restricted to FIFO and can be extended to other policies such as Fixed Priority Preemptive Scheduling.

4.3 Schedulability analysis and scheduling synthesis

The CPAL execution engine possesses mechanisms to monitor and record at run-time the execution time of the processes. This feature can be taken advantage of by the designer to estimate the Worst-Case Execution Times of the processes making up the application. Since the workload submitted to the runtime environment is statically defined and fully characterized, it is possible to derive a schedulability analysis for a set of CPAL processes. We developed in [Alt15b] two schedulability analysis: an exact test based on simulation and an approximate test based on the schedulability test for non-preemptive scheduling with offsets [Pe05]. A feasibility test via simulation requires simulation up to twice the hyperperiod, which may be infeasible in many situations. In the latter case, we have to resort to the approximate schedulability test.

We believe that significant progresses in terms of development time and correctness can be achieved by further automating the design process. In the timing dimension, this can for instance be done by synthesizing a feasible scheduling solution with this two-steps approach developed in [Al15c]:

- the developer states the permissible timing behavior of the system using a declarative language for the specification of non-functional timing properties,
- a system synthesis step involving both analysis and optimization (e.g., periods and offsets) then generates a scheduling solution which at run-time is enforced by the execution environment.

The interested reader can refer to [Al15c] for a more comprehensive discussion on scheduling synthesis in the CPAL framework.

5 Related work

A motivation behind CPAL is that general purpose programming languages abstract away timing considerations, and non-functional properties at large. They also lack the domain-specific constructs that are needed to speed-up the development, facilitate the re-use and the understanding of real-time embedded software.

Synchronous programming models, be they functional like Lustre [Ha91] and Signal [Be91] or imperative like Esterel [Bo91], propose an effective and sound answer to facilitate the correct design of reactive systems made up of concurrent tasks. Thanks to their formal semantics, they have brought major progresses to the development of safe embedded systems over the last 30 years, and are certainly well suited in some application domains such as safety-critical systems. However, the learning curve is steep for programmers used to more conventional programming languages. In addition, the complexity of the formalisms that need to be understood or manipulated is a hindrance to their adoption by the practitioner. Also the programming style and the abstractions offered by the languages do not fit all problems and programmers. In many cases, we also believe that more lightweight and less demanding programming models are equally able to guarantee the necessary timing predictability without over-constraining the design and development.

Amongst the synchronous approaches, CPAL has been inspired by Giotto [He03] which is a time-triggered architecture language. Indeed, several mechanisms available in Giotto are re-used such the task activation models (e.g., periodic process, guarded executions). In that respect, the current task activation model of CPAL can be seen as purely time-triggered. However, on the contrary of CPAL, neither Giotto defines a runtime environment nor is it a programming language to express functional behavior. In addition, although we could imagine implementing an execution semantics in CPAL compliant with Giotto, CPAL currently relaxes the programming model of Giotto in several ways:

- No system-wide mode change mechanisms as in Giotto are defined in CPAL, which supports mode changes through guarded executions at the process-level.
- Although the order of observable events is deterministic in CPAL, the outputs of a process are not produced at a predetermined point in time. They may be several output times that may be subject to variations depending on the actual execution times of the processes, but the interval where they happen can be bounded by schedulability analysis.
- Unlike in Giotto, input ports of a process are read at the actual start of the process execution and not upon (or before) its release time. The process thus works on the most recent data.
- In CPAL it is possible to explicitly re-read an I/O-mapped variable or to perform several writings to an output port during the execution of a process (e.g., to drive serial communication by bit-banging, send segmented messages, etc). This is done though the dedicated `syncIO()` function.

A more recently proposed architecture description language is Prelude [Fo09,Fo10] which extends synchronous approaches, such as Lustre, to facilitate the development of multi-rate applications with complex communication patterns between tasks. Prelude builds on the formal synchronous model to offer powerful operators (e.g., over and under-sampling) to define the flows of data between functions potentially operating at different rates. Prelude is able to perform correctness checks that ensure that the program has a deterministic semantics. Then, the Prelude compiler translates the program into a set of communicating real-time tasks scheduled in such a way as to meet the timing constraints. Like Giotto, Prelude is not a programming language to define the actual functional behavior of the tasks, neither is it an execution platform. Preliminary experiments on examples such as the flight application software in [Fo10] suggest that most of the semantics of Prelude programs can be captured in CPAL. Future work will be devoted to assess the feasibility of transforming CPAL programs into Prelude programs in order to take advantage of the data-flow verification framework readily available within Prelude.

6 Conclusion

The CPAL programming language and associated toolset is a model-driven development flow aimed at the development of timing-predictable embedded systems. Our priority in the language design has been to favor simplicity, user-friendliness and expressive power both in the functional and non-functional dimensions. In particular, CPAL provides language abstractions needed to define real-time applications and argue about the timing correctness on a high-level.

In that way, CPAL is a contribution towards addressing what Thomas Henziger called the grand challenge in embedded software design [He08]: "offering high-level programming model that exposes the execution properties of a system in a way that permit the programmer to express desired reaction and execution requirements, permits the compiler and run-time systems to ensure that these requirements are satisfied". CPAL provides a programming model, easier to handle for most programmers than synchronous approaches, which aims at ensuring timing-predictability instead of time-determinism which is over-constraining in many real-time applications.

CPAL has been already successfully used to answer several industrial problems (Al15, Ci16, Se15), as well as to teach MDD. Upcoming releases of the development environment and the CPAL interpretation engine will gradually offer an integrated support for off-line and on-line verification activity.

7 References

[Al15] S. Altmeyer, N. Navet, L. Fejoz, “Using CPAL to model and validate the timing behaviour of embedded systems”, WATERS Workshop, July 2015. Available at http://hdl.handle.net/10993/21250.

[Al15b] S. Altmeyer, N. Navet, “The case for FIFO scheduling”, technical report from the University of Luxembourg, to appear, November 2015.

[Al15c] S. Altmeyer, N. Navet, “Towards a declarative modeling and execution framework for real-time systems”, First IEEE Workshop on Declarative Programming for Real-Time and Cyber-Physical Systems, San-Antonio, USA, December 1, 2015.

[An12] M. Antoni, “Formal validation method and tools for computerized interlocking system”, 18th International Symposium on Formal Methods (FM 2012), Industry day, August 27-31, 2012. Slides available at http://fm2012.cnam.fr/fm2012/ID2012-Marc-Antoni.pdf.

[Be91] A. Benveniste, P. Le Guernic, C. Jacquemot » Synchronous programming with events and relations: the Signal language and its semantics”, Science of Computer Programming, 16(2), 1991.

[Bo14] F. Boniol, V. Wiels, “The landing gear system case study”, pp1-18, Proc. ABZ 2014, 2014.

[Bo91] F. Boussinot, R. de Simone, “The Esterel language”, Proc. IEEE, 79(9), 1991.

[Br04] A. Brown, “An Introduction to Model Driven Architecture – Part1: MDA and today’s systems”, IBM technical library, 2004. Available at http://www.ibm.com/developerworks/rational/library/3100.html.

[Ci16] L. Ciarletta, L. Fejoz, A. Guenard, N. Navet, “Development of a safe CPS component: the hybrid parachute, a remote termination add-on improving safety of UAS”, to appear at ERTSS2016, Toulouse, January 2016.

[DA14] S. Di Alesio, S. Nejati, L. Briand, A. Gotlieb, “Worst-Case Scheduling of Software Tasks - a Constraint Optimization Model to Support Performance Testing”, 20th International Conference on Principles and Practice of Constraint Programming, 2014.

[Fe15] L. Fejoz, N. Navet, “The CPAL Programming Language – an Introduction”, available at http://www.designcps.com, 2015.

[Fo09] J. Forget, “Un Langage Synchrone pour les Systèmes Embarqués Critiques Soumis à des Contraintes Temps Réel Multiples”, Phd Thesis in Computer Science from the University of Toulouse, 2009.

[Fo10] J. Forget, F. Boniol, D. Lesens, C. Pagetti, “A real-time architecture design language for multi-rate embedded control systems”, 2010 ACM Symposium on Applied Computing (SAC '10), pp527-534, 2010.

[Ha91] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, “The synchronous data-flow programming language LUSTRE”, Proc. IEEE, 79(9), 1991.

[He03] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, “Giotto: A time-triggered language for embedded programming”, in Proceedings of the IEEE, vol. 91, n°1, pp 84–99, 2003.

[He08] T. A. Henzinger, “Two challenges in embedded systems design: predictability and robustness”, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 366(1881):3727–3736, 2008.

[Hu03] J. Huang, J. Voeten, A. Ventevogel and L. Van Bokhoven, “Platform-independent Design for Embedded Real-Time Systems”, in Proceedings of FDL'03, pp. 318-329, 2003.

[Ma03] F. Maraninchi and Y. Rémond, “Mode-Automata: a new Domain-Specific Construct for the Development of Safe Critical Systems”, Science of Computer Programming, n°46, pp, 219-254, 2003.

[Mo11] A. Monot, N. Navet, and B. Bavoux, “Impact of clock drifts on CAN frame response time distributions”, in 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011), 2011.

[Mo12] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, “Multisource software on multicore automotive ECUs combining runnable sequencing with task scheduling”, IEEE Transactions on Industrial Electronics, 59(10):3934–3942, Oct 2012.

[Na14] N. Navet, “Lean Model-Based Design for Cyber-Physical Systems – the case for Model Interpretation”, invited talk at ABB Corporate Research, Basel, Mai 14, 2014.

[Pe05] R. Pellizzoni, G. Lipari, “Feasibility analysis of real-time periodic tasks with offsets”, Real-Time Systems, 30(1-2):105–128, May 2005.

[Se15] J. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, “Formal Analysis of the Startup Delay of SOME/IP Service Discovery”, DATE 2015, Grenoble, France, March 9-13, 2015.

[Tr09] T. Trew, “Creating Embedded Platforms with MDA: Where's the Sweet Spot”, slides presented at ECMDA-FA, 2009.

[Wa15] Community Forum on Tools and Benchmarks for Real-Time Systems, http://ecrts.eit.uni-kl.de/forum, 2015.

Making Modeling Assumptions an Explicit Part of Real-Time Systems Models

Pierre de Saqui-Sannes[1] and Ludovic Apvrille[2]

[1] Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse Cedex 04, France
pdss@isae-supaero.fr
[2] LTCI, CNRS, Télécom ParisTech, Université Paris-Saclay, Biot, France
ludovic.apvrille@telecom-paristech.fr

Keywords. Model-Based System Engineering, Modeling Assumption, SysML.

1 Introduction

Facing parallelism, determinism, and timelines problems, a real-time system designer attempts to master design complexity by making abstractions and creating models. For instance, somebody modeling a pressure controller may assume the pressure sensor and the alarm never fail. Beyond this simple example, it clearly turns that a real-time system model is valid for a precise set of assumptions. An explicit knowledge of these assumptions is an asset to make the model easy to share. Moreover, this knowledge contributes to remove ambiguities.
Surprisingly, little work has been published on the necessity to make modeling assumptions a full part of a system model. Conversely, this paper advocates for an explicit inclusion of modeling assumptions into the system model and discusses solutions in the context of the OMG's System Modeling Language (SysML [1]). The solution illustrated on the definition of so-called "Modeling Assumptions Diagrams" is not restricted to SysML and may be reused, e.g., for UML or AADL. An important issue discussed by the paper is the versioning one. Indeed, our approach proposes to explicitly set or release modeling assumptions along an incremental modeling process.
The paper is organized as follows. Section 2 sketches a process that explicitly include modeling assumptions definition. Section 3 introduces Modeling Assumption Diagrams (MADs fort short). The TTool tool [4] adds MADs to SysML. Section 4 discusses a case study: a UAV platform for autonomous navigation across buildings. Section 5 surveys related work. Section 6 concludes the paper.

2 Process

The SysML standard defines a notation, not a method. We thus need to associate a process with the SysML language. The one described below is supported by the free (i.e., "libre") TTool tool [4]. The latter supports several UML/SysML profiles. The SysML framework considered in this paper is called AVATAR [2]. TTool supports the Modeling Assumption Diagram presented in this paper as well as the whole AVATAR process described below.

1. *Assumptions.* One or several Modeling Assumption Diagrams captures assumptions linked with the system under design and its environment.
2. *Requirement capture.* A Requirement Diagram hierarchically organizes safety and security requirements with explicit notion of refinement and derivation of technical requirements from logical ones.
3. *Analysis.* One or several Use-Case Diagrams identify the main functions and services offered by the system. The use cases are documented by scenarios (sequence diagrams) and flow charts (activity diagrams).
4. *Design.* An AVATAR design defines the architecture of the system and the behaviors of the block instances the architecture is made up of. The architecture and the behaviors are respectively modeled by a Block Instance Diagram and State Machine Diagrams.
5. *Design validation.* To check an AVATAR design against errors and its expected requirements, the model itself is analyzed using simulation and verification tools directly accessible from TTool with a push-button approach. Available techniques include early debugging using step-by-step and random simulation, safety property verification using invariants search techniques or UPPAAL [5], and security flaws detection using Proverif [3]. Both simulation and safety/security verification are driven from TTool and the results are displayed on the SysML model itself or at a high level that does not oblige the SysML model designer to investigate an intermediate formal code.
6. *Design prototyping.* Model transformations techniques enable generation of C/POSIX code from AVATAR design diagrams. Linking TTool to SocLib allows one to prototype the generated code in complex platforms [21].

3 Modeling Assumptions Diagrams

A Modeling Assumption Diagram captures a set of assumptions, with their respective attributes, as well as the interconnections between assumptions.

3.1 Assumption Nodes

There exists two types of assumption nodes, respectively stereotyped by << *System* >> and << *Environment* >>. The former (resp. the latter) is used for assumption linked to the system itself (respectively the environment of the system). Both System and Environment assumption nodes share the following list of attributes:

– *Durability.* Because modeling is usually an incremental process, a modeling assumption does not necessarily apply to all the versions of the model. For instance, a system model that first ignores maintenance may be upgraded to make the user interface interact with a supervisor in charge of maintenance. The assumptions associated with the maintenance receives a *Durability* attribute equal to *temporary* (as opposed to *permanent*). Another attribute states whether the assumption remains *applied* in the current version of the model or whether it it has been *alleviated* in the current or in a previous version of the model.

– *Source.* An assumption may originate from the *end-user*, the *stakeholder* or the *creator* of the model, respectively.
– *Status.* The status gives an information on whether an assumption is totally *applied* or partially applied (*aleviated*) in the model.
– *Scope* may take one or several of the following values:
  - *Language* addresses either a syntactical restriction, a semantics-related limitation, or a limited expression power of the modeling language.
  - *Tool* denotes a limitation of the tool (e.g., an unsupported construct of the modeling language, a limited resource, e.g., memory).
  - *Modeling activity* relates the way the company uses the modeling language and the tools, e.g., by forbidding non deterministic constructs.
  - *Verification* deals with the verification limitation introduced by specific underlying verification toolkits. Examples include state space explosion problem, undecidability, and non support of temporal constraints.

3.2 Relations between assumptions

The metamodel also defines relations that link pairs of assumptions.

– *Containment.* A complex modeling assumption is split up into two or several elementary assumptions. Its semantics is similar to the one of the requirement containment in SysML.
– *Versioning.* It links two assumption nodes a and b. The relationship $<< versioning >>$ going from a to b, and qualified with $\{x \rightarrow y\}$ means that assumption a applies until version number x and is superceeded by assumption b starting at version number y.
– *Relation between an assumption and a reference to a modeling element.* The $<< impact >>$ relation states that the referenced element at the destination of the link is directly impacted by the assumption at the origin of the link. The referenced elements can either be a diagram, or a modeling element, e.g., a block, the state of a state machine, etc.
– *Composition relations between a reference to a diagram and references to elements.* The composition relation can be used to make more explicit the fact that a diagram contains impacted elements.

4 Case Study

A UAV system serves as case study. All the diagrams have been edited using the latest release of TTool, which supports MADs.

4.1 Informal specification of the UAV

In the incoming years, micro-drones could play a key role in our society, namely the role of assistant, in particular in the scope of disasters. However, their manipulation currently requires specific skills (flying skills, mission-related skills) that rescue teams are not ready to invest on. Thus, drone autonomy is a research topics on which Telecom ParisTech and EURECOM have been working on for several years in the *drone4u* project [22]. In particular, drone4u studies how to perform autonomous drone navigation in harsh conditions, in particular inside buildings. Drone4u has already investigated three scenarios of autonomous navigation:

1. Following and understanding marks, e.g., a red line located on the floor, indication marks on doors.
2. Analyzing the environment (obstacles, etc.) with image-based processing techniques (3D reconstruction).
3. 3D reconstruction with human assistance in order to go through obstacles that a drone cannot handle on its own, e.g., entering in a room when the entrance door is closed.

4.2 Modeling assumptions

The first model considers only the first scenario. The corresponding MAD is given in Figure 1. The main assumption concerns the signs that are necessary to navigate, in particular the red line located on the floor. *DroneDesign1* handles that line-based guidance, and implements it, in particular, with a block named *LineRecognitionAlgorithm*, see Figure 2.

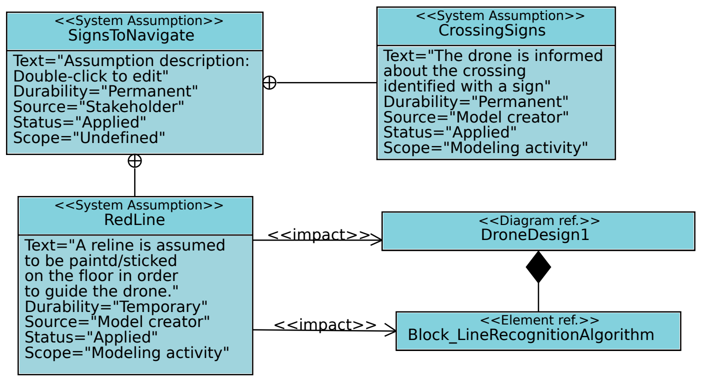


Fig. 1. Modeling Assumption Diagram - version 1

In scenario 2, the drone does not follow a line anymore, but uses images from the camera to detect paths to follow and obstacles. The *RedLine* assumption is thus now deprecated and replaced by a new assumption named *ThreeDRecognition* (see Figure 3). The design named *DroneDesign2* handles that new assumption by two means. First, the introduction of a new block named *ThreeDRecognitionAlgorithm* and second, the modification of the block *Camera*.

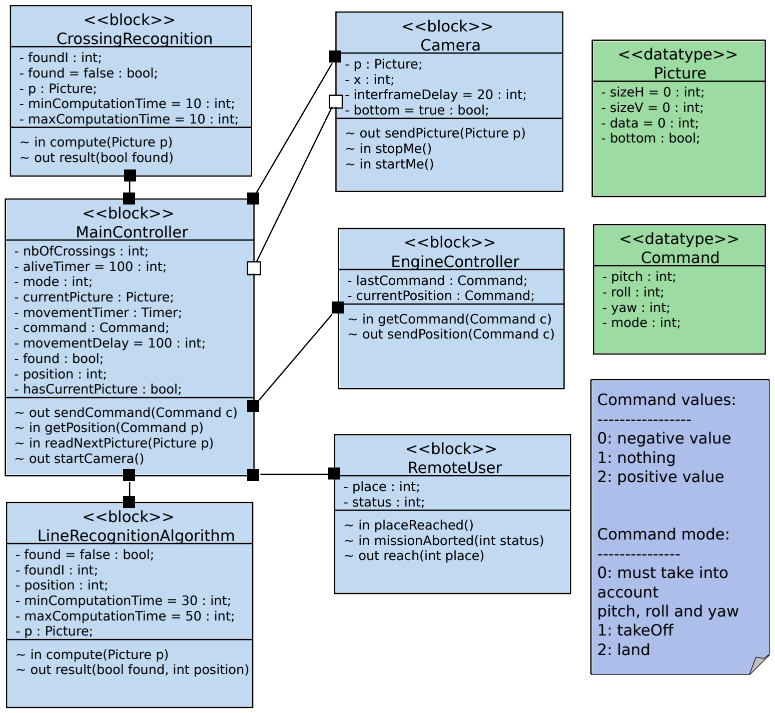


Fig. 2. Avatar block instance diagram - Drone Design 1

At last, scenario 3 takes into account the fact that the drone can go through doors. It means that scenario 2 was assuming there was no door (but negative assumption are more rarely expressed), and scenario 3 now assumes that a person must assist the drone to go through doors. A new MAD is thus used to express those new assumptions (see Figure 4). In particular, the temporary "NoDoor" assumptions applies until version 2, but no more in version 3, in which the "FollowingPersons" applies.

The design *DroneDesign3* (see Figure 5) handles the *FollowingPersons* assumption. Modifications to meet that assumption impact two blocks: *MainController* and *RemoteUser*. The two blocks are enhanced with signals (e.g., *DoorDetected*, *DoorHandled*), attributes and their state machine diagram can handle the new situation (going through doors).

4.3 Discussion and limitations

The case study demonstrates the facility to manage assumptions for different versions of the same system: assumptions, as well as their main characteristics can easily be captured, and the versioning is explicit in the diagram. The impact of modifying assumptions can be traced both at diagram level, and at modeling elements level.

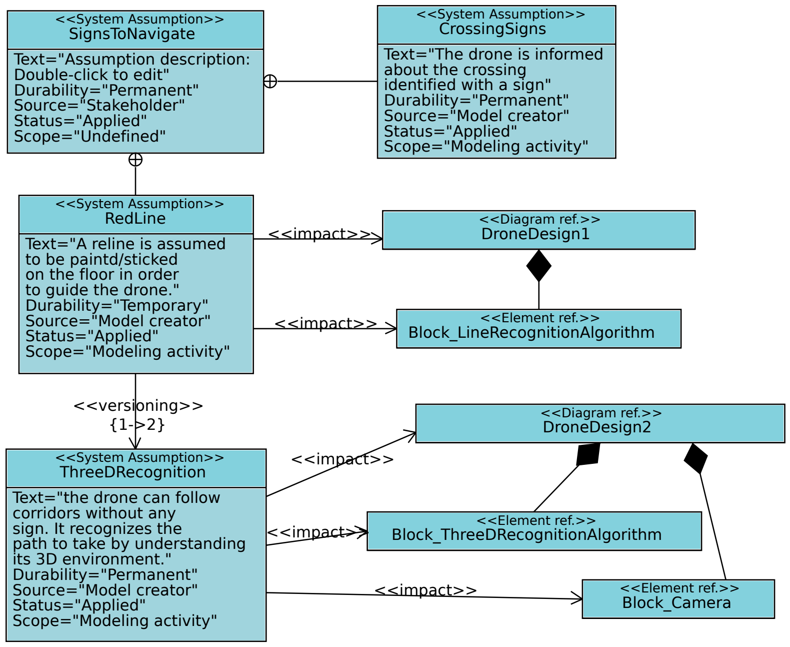


Fig. 3. Modeling Assumption Diagram - version 2

Consistency checking inside of Modeling Assumption Diagrams, and with regards to referenced elements, has also been defined. For example, concerning versioning, an assumption modeled as deprecated in version x, should not itself deprecate another assumption in version x. For referenced elements, if an element, e.g., a block, meets a new assumption in version x, it should be different from the same block in version y with $y < x$. TTool already partly performs these coherency checking, which are necessary for correctly handling assumptions traceability and interest. Similarly, helping the designer to rework the MAD while improving the other diagrams is a point that we intend to address in a near future.

Currently, one needs to open a MAD to see which elements are impacted ($<< impact >>$ relations) by given assumptions. It could also be interesting to vizualize directly on design diagrams the assumptions linked to elements, e.g., blocks, states of state machines. For example, when passing the mouse pointer over a given element, TTool could display in a popup window all the related elements. Table-oriented views could also be used to better trace assumptions and modeling elements, in the same way as TTool can currently make it for requirements.

Last but not least, TTool can perform simulation and formal verification of safety and security properties on designs. However, TTool cannot (yet) use the modeling assumption diagrams to evaluate which parts of the design have

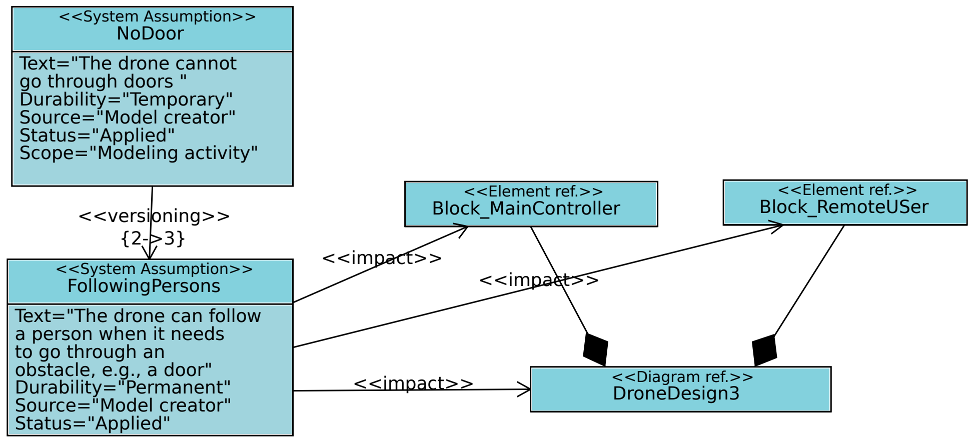


Fig. 4. Modeling Assumption Diagram - version 3

evolved - using the $<<$ *impact* $>>$ relation - in order to restrict the proof to the properties that still have to be proved again in a new design version.

5 Related Work

A survey of the literature indicates that numerous authors (e.g. [14] [15]) share the following idea: specifying the assumptions is a system-engineering task.

[14] adds that "model assumptions must be stated explicitly". [17] says "Some authors treat a list of assumptions as if were a conceptual model. It is not; but a conceptual model should include a list of assumptions". [18] further states "We want to have an argument that increases our confidence that the model represents the system correctly". Therefore we document some of the modeling decisions in form of a list of the systems assumptions made while modeling. Given a real-time system boiled down to a controller and an environment termed as "plant", [18] categorizes assumptions that respectively apply to the system's components, to the properties related to the mechanical, electrical and software that form the aspects of a system.

[16] categorizes the artifacts of conceptual modeling into "knowledge acquisition" and "model abstraction". The former demands to acquire knowledge about the real world and to derive a system description. The latter abstracts a conceptual model from a system description, which implies specifications are made.

[10] further categorizes assumptions into technical assumptions, organizational assumptions and managerial assumptions. Examples of technical assumptions include programming languages as well as database and operating systems. Conversely, the paper advocates for a clear distinction between the system's environment (in terms e.g. of network connection and operating system) and the modeling or programming language used to design and develop the system. In

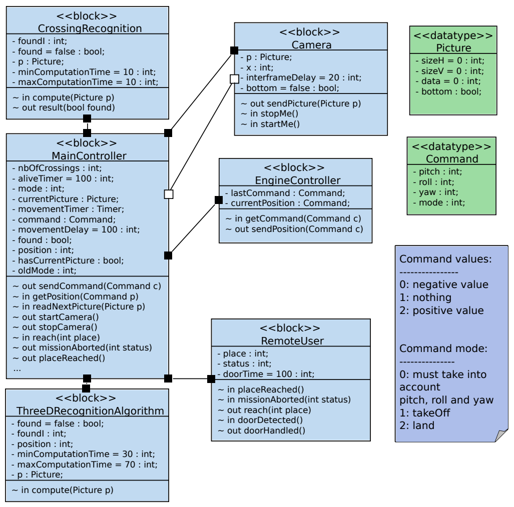


Fig. 5. Avatar block instance diagram - version 3

terms of organizational assumptions, our paper does not cover all the issues (development team, workflow, company standard) addressed by [10], assuming the method associated with AVATAR is compatible with the practice in use in the company. Also, the paper does not address managerial assumptions. Finally, [10] associates assumptions to architectural models and therefore to design decision where the paper also enables to link assumptions to analysis and behavioral diagrams.

[17] defines an assumption and the way a system uses it. [17] terms as "referent" the model or system component that the assumption is about; it compares to the "impact" relation in MADs. Also, [17] terms as "scope" a description of which parts the assumption refers to; it compares to the "impact" relation in MADs. [17] also distinguishes between the system and its components.

Assumptions management is a key issue for versioning. [17]'s classification on change management in the context of requirement diagrams may be a source of inspiration for classifying the operations associated with a $<< versioning >>$ relationship: (1) New assumption added, (2) Existing assumption removed, (3) Part remove from an assumption, and (4) New part added to an assumption. Whatever the operation, the effect of applying a new modeling assumption at the end side of a $<< versioning >>$ relation may easily span over several diagrams. Evaluating that effect may rely on a dependency graph computed from the relations that link assumptions to diagram and diagram elements.

The tutorial in [8] mentions "List requirement and assumptions" as an early stage of a process applied to a distiller. The authors use the "Rationale" keyword to characterize an assumption described in a comment. The latter is attached to a requirement. Nothing is said about the traceability of the assumption made as a comment.

[20] addresses assumptions in a UML/SysML context. The paper proposes to extend UML/SysML with contracts. A contract associated with one component is a pair (assumption, guarantee) where an assumption is abstraction of the component's environment behavior and the guarantee is an abstraction of the component's behavior given that the environment behaves according to the assumption.

[19] also addresses assumptions in a SysML context. Given safety requirements arise from assumptions about the system's context, [19] introduces the concept of environmental assumptions and describes them by two means: SysML parametric diagrams for continuous properties of a hardware entity, and OCL constraints for discrete properties of a software entity. Besides standards, customers and domain experts, environmental assumptions belong to the list of sources the requirements come from.

6 Conclusions

A model is hard to understand and transmit as long as one ignores the assumptions made by the engineers who elaborate that model. Therefore, the authors of the paper strongly advocate for an explicit inclusion of modeling assumptions inside the model. The paper implements the idea in the context of AVATAR, the real-time variant of SysML that is supported by the TTool tool. Adding Modeling Assumption Diagrams to AVATAR is the main purpose of the paper. A MAD defines a set of modeling assumptions nodes that describe and categorize the assumptions made to elaborate the model. A complex assumption may be split up into elementary ones. A versioning arrow between two assumptions enables to achieve versioning. Also, an assumption may be linked to a diagram it has an influence on. Precision may be added on the way diagram elements are impacted by the diagrams themselves influences by one or several assumptions. The autonomous UAV that serves as case study is a real UAV
(https://www.youtube.com/watch?v=tamYpmGvzRw).
All the diagrams presented in the paper have been edited using TTool.

The concept of Modeling Assumption Diagram could be applied to OMG's SysML and to UML, as well as to UML profiles in general. The concept of versioning deserves further investigations in the view of optimizing the simulation and formal verification activities that may applied on AVATAR models.

References

1. OMG, "System Modeling Language, specification v1.4", http://www.sysml.org/, 2015.
2. Ludovic Apvrille, Pierre de Saqui-Sannes, "Static analysis techniques to verify mutual exclusion situations within SysML models", 16th International System Design Languages Forum, Montreal, Canada, Juin 2013.

3. Ludovic Apvrille, Yves Roudier, "SysML-Sec: A SysML Environment for the Design and Development of Secure Embedded Systems", Proceedings of the INCOSE/APCOSEC 2013 Conference on system engineering, Yokohama, Japan, September 8-11, 2013.
4. Ludovic Apvrille, "TTool, a free UML/SysML modeling software", http://ttool.telecom-paristech.fr/
5. http://www.uppaal.org/
6. OMG, "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems V1.1", http://www.omgmarte.org/, 2011
7. "SoCLib, an open platform for virtual prototyping of multi-processors system on chip (MP-SoC)", http://www.soclib.fr/
8. S. Friedenthal, A. Moore, R. Steiner, OMG Systems Modeling Language, tutorial, 19 June 2008, http://www.uml-sysml.org/documentation/sysml-tutorial-incose-2.2mo
9. R.D. King, C.D. Turnitsa, The Landscape of Assumptions, SpringSim '08, Proceedings of the 2008 Spring simulation multiconference, Pages 81-88
10. Lago, P. and van Vliet H., Explicit assumptions enrich architectural models, 27th Int. conference on Software engineering, St. Louis, MO, USA, 2005.
11. Towards Requirements Aware Systems: Run-time Resolution of Design Assumptions, 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lawrence (KS), USA, 2011.
12. J. Marincic, A. Mader, A. Wieringa, Capturing Assumptions while Designing a Verification Model for Embedded Systems, January 2007.
13. Ludovic Apvrille, Alexandre Becoulet, "Fast and Multi-platform Prototyping of Embedded Systems from UML/SysML Models", The 14th edition of the Sophia Antipolis MicroElectronics Forum (SAME'2011), Sophia Antipolis, France, Oct. 12-13, 2011.
14. H. Kopetz, P. Puschner, RT System Modeling, TU Wien, May 2013, http://ti.tuwien.ac.at/cps/teaching/courses/real-time-systems/slides/rts03_rt_model.pdf
15. F. Kordon, J. Hugues, A. Canals and A. Dohet, Embedded Systems: Analysis and Modeling with SysML, UML and AADL, ISTE / Wiley, May 20, 2013, ISBN-13: 978-1848215009.
16. K. Kotiadis, S. Robinson, Conceptual Modeling: Knowledge Acquisition and Model Abstraction, Winter Simulation Conference, Miami, FL, USA, December 2008.
17. R. D. King, C. D. Turnitsa, The Landscape of Assumptions, Proceedings of the 2008 Spring simulation multiconference, San Diego, CA, USA, Pages 81-88.
18. J. Marincic, A. Mader, R. Wieringa, Classifying Assumptions Made During Requirements Verification of Embedded Systems. In: 14th International Working Conference on Requirements Engineering: Foundation for Software Quality, REFSQ 2008, 16-17 June 2008, Montpellier, France (pp. pp. 141-146).
19. Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, Thierry Coq, A SysML-Based Approach to Traceability Management and Design Slicing in Support of Safety Certification: Framework, Tool Support, and Case Studies, Journal of Information and Software Technology, Volume 54 Issue 6, June, 2012, Pages 569-590.
20. Iulia Dragomir, Iulian Ober, Christian Percebois, Integrating verifiable Assume/Guarantee contracts in UML/SysML, ACESMB@MoDELS, volume 1084 of CEUR Workshop Proceedings, CEUR-WS.org, (2013)
21. Daniela Genius, Ludovic Apvrille, Virtual Yet Precise Prototyping: An Automotive Case Study, Proceedings of the ERTS'2016, Toulouse, France.
22. Ludovic Apvrille, Jean-Luc Dugelay "Drone4u", http://drone4u.eurecom.fr

An Architecture-Led Safety Analysis Method

Peter H. Feiler
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
phf@sei.cmu.edu

David P. Gluch
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
dpg@sei.cmu.edu

John D. McGregor
School of Computing
Clemson University
Clemson, SC 29634 USA
johnmc@clemson.edu

Abstract

Safety-critical systems require specific development and evaluation activities in the software development life cycle to ensure that the product is safe. Some of these activities are aggregated into comprehensive safety engineering practices, which are standardized within an industry, such as Aerospace Recommended Practice (ARP) 4761 in the aircraft industry. These techniques focus on individual component failures and reliability. More recent techniques such as the Systems-Theoretic Process Analysis (STPA) go beyond reliability of individual components to consider the interactions among the components. In this paper we present the Architecture-Led Safety Analysis (ALSA) method that is part of the Architecture-Led Safety Engineering practice. ALSA combines the development and analysis of at least a partial architecture model using notations such as the Architecture Analysis and Design Language, its Error Model Annex, and existing ARP 4761 and ARP 4754A practices such as Functional Hazard Assessment, Preliminary System Safety Assessment, and System Safety Assessment as well as the emerging technique of STPA. This work contributes an illustration of using ALSA to analyze a Full-Authority Digital Engine Controller. The method is supported by the Open Source Architectural Tool Environment and has been piloted on an industrial-strength example.

Keywords: safety analysis; architecture-led; error model

Introduction

Safety-critical systems require specific development and evaluation activities in the software development life cycle to ensure that a system is safe. Some of these activities are aggregated into comprehensive safety engineering practices, which are standardized within an industry. Standards such as Aerospace Recommended Practice (ARP) 4761 in the aircraft industry define very specific techniques for assessing the safety of life-critical systems [1]. These techniques focus on individual component failures and their associated reliability. More recent techniques such as the Systems-Theoretic Process Analysis (STPA) go beyond methods based on the reliability of individual components to consider the interactions among the components [2].

The increasing complexity of current embedded systems is driving the need for increasing automation of all activities, including safety analysis. The goal of our research is to create a safety engineering practice that blends existing safety analysis techniques with an error ontology and a model-based architecture representation of the system and its operational context that will more effectively and efficiently discover hazards in the interactions among system components. The practice enables increased automation to support iterative development. Our technique differs from some existing practices as described in Chen and Biehl that use EAST-ADL [3–5]. EAST ADL focuses on supporting system engineering mapping functional architectures onto hardware. It does not support modeling of the software architecture. In contrast, AADL allows modeling of system and software architectures. By supporting safety analysis across both, it addresses hazard contributions by software. There is plenty of recent evidence that software has become a major source of hazards; thus, it is crucial to extend safety analysis into the software system architecture.

In this paper we present the Architecture-Led Safety Analysis (ALSA) method and demonstrate its use. ALSA is part of an Architecture-Centric Virtual Integration Process (ACVIP) within a comprehensive, architecture-led systems engineering practice [6]. Figure 1 shows steps in the ACVIP, with the ALSA steps in boldface. We apply these process steps repeatedly down the hierarchy of subsystems. The diagram conveys that the ACVIP and ALSA steps are closely coupled and involve iteration and concurrency of activities.

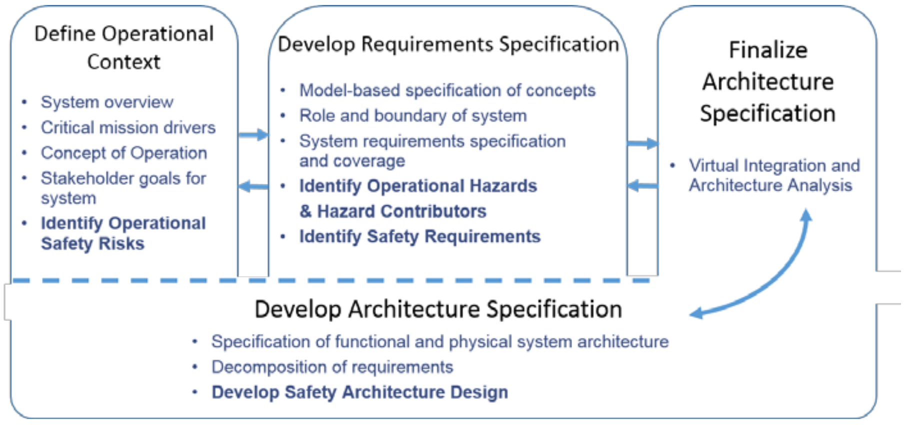

Define Operational Context
System overview
Critical mission drivers
Concept of Operation
Stakeholder goals for system
Identify Operational Safety Risks
Develop Requirements Specification
Model-based specification of concepts
Role and boundary of system
System requirements specification and coverage
Identify Operational Hazards & Hazard Contributors
Identify Safety Requirements
Finalize Architecture Specification
Virtual Integration and Architecture Analysis
Develop Architecture Specification
Specification of functional and physical system architecture
Decomposition of requirements
Develop Safety Architecture Design

Figure 1: Process steps for ACVIP (lightface) and ALSA (boldface).

Our practice combines features of several existing safety engineering practices, model-based requirements analysis, and architecture design practices. It incorporates development and analysis of at least a partial architecture model using notations such as the Architecture Analysis and Design Language (AADL) [7]. The method incorporates existing ARP 4761 and ARP 4754A practices such as Functional Hazard Assessment (FHA), Preliminary System Safety Assessment, and System Safety Assessment. Our method also uses techniques similar to those in STPA, including a specially constructed set of guidewords. The unique contribution of our work is the use of an architecture representation of the system and its operational context and an error ontology to support hazard identification. An early version of this method was validated by its use in the System Architecture Virtual Integration technique [8].

We create the architecture representation, which supports our practice, using AADL and elaborate it over multiple iterations of analysis and design. Several annexes to the core AADL have been defined and we particularly use the error annex, referred to as EMV2 (error model version 2), in this safety analysis method. EMV2 supports expressing the error behavior of a component and its subcomponents using a state machine construct linked to the component's ports. Component connectors between ports and error flows through the connectors allow the designer to explicitly plan error propagations through the system. Work around, restart and other error recovery strategies can be modeled and analyzed. EMV2 defines the error ontology that is a center piece of the safety analysis method. The ontology provides guide words that are used to help locate safety hazards, which are added to the error model as properties that can be used to produce hazard analysis reports.

An architecture model created using the process in the rest of this paper supports safety analyses at user-selectable levels of detail. Certain analyses are conducted on an instantiation of the selected component including any subcomponents of the selected component. The algorithms are defined to accept the scope of the selected entity and to apply the analysis technique within that scope. The highly extensible OSATE predefines analyses such as FHA but as an open source tool based on Eclipse any analysis can be added to the tool set.

The practice is composed of four steps that we apply in a top-down manner to the system hierarchy. Feedback from lower levels drives the next iterations with new information. The individual steps interface with the encompassing ACVIP practice and inform other integration activities. The four steps are

1. Identify Operational Safety Risks: This step involves identifying operational system-level accidents, incidents, and contributory system-level hazards by considering the context in which the system is engineered and operates as well as its interfaces with the environment. This step requires significant stakeholder engagement, especially safety engineering, operational, and mission expertise.
2. Identify Operational Hazards and Hazard Contributors: Step 1 helps to establish a minimal system architecture, associating identified accidents with various aspects of the system. The error ontology defined in the Error Model Annex of AADL provides guidance in identifying hazards and their contributors.
3. Identify Safety Requirements: The sources of error are used to define safety requirements that mitigate the identified hazards. While designated as a distinct step, safety requirements can be defined concurrently with hazard identification.
4. Develop Safety Architecture Design: A safety architecture is designed based on the safety requirements identified in the previous step. The safety architecture merges the safety aspects of the product into the overall product architecture.

The ALSA practice is supported by the Open Source Architectural Tool Environment (OSATE) [9]. It is conducted in coordination with the general systems requirements definition and initial architecture design activities in an iterative, incremental development approach. The architecture representation includes information about nominal and error flows in the system. These flows are analyzed end to end to identify hazards that may appear only when complex interactions among multiple components are considered together. As hazards are discovered, the architecture representation is annotated with information about the hazards, making this information available in future iterations of analysis as the system definition evolves.

The example in the next section illustrates elements of the ALSA approach but is not intended to represent a comprehensive safety assessment. In practice, experts utilize these techniques to develop the technical and safety aspects of the systems they analyze. In presenting the ALSA process, we assume that readers are familiar with the AADL, the AADL Error Model Annex (EMV2), and their application [10–12].

Example

This is an example application of the ALSA safety process to a representative Full-Authority Digital Engine Controller (FADEC) system. For our purposes, we address only the operating process aspects of the broader system-level theoretical framework presented in Figure 1. The example focuses on the fuel flow control aspects of the system as shown in Figure 2 and taken from [13]. The design presented here is illustrative, does not represent any specific or operational FADEC system, and is not intended for implementation. Nominally, a safety analysis is conducted for the complete aircraft, but in this problem (i.e., the system) we focus on the aircraft engine. A technical report is in preparation and will provide many details that are omitted here due to space limitations. We apply each of the four steps in the safety process and explain the interactions among the steps.

Identify Operational Safety Risks

This initial step interfaces with the encompassing ACVIP and defines the operational safety context for the system as part of the ACVIP context-definition activity. In this step, we identify safety risks (accidents, incidents, and top-level operational hazards). As noted earlier, the specific procedures, techniques, and outputs depend on the preferences and norms of an organization. In some cases, certifications require specific practices.

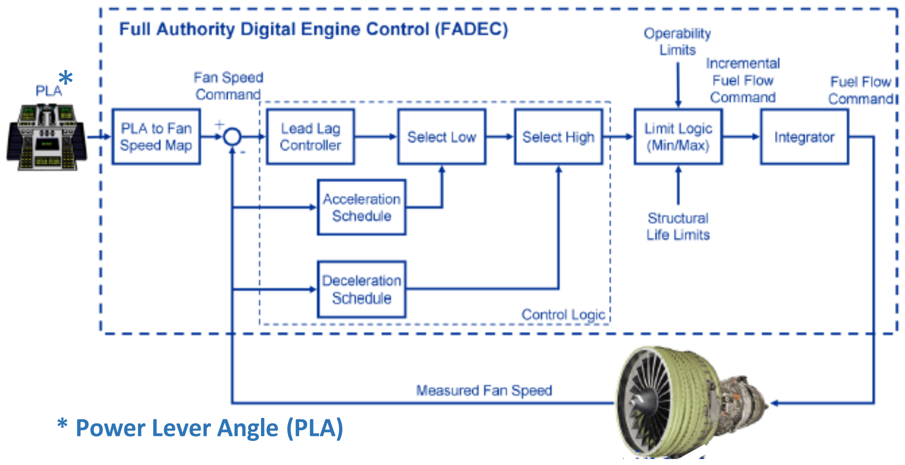


Figure 2: FADEC fuel flow control example [data adapted from 10].

Various techniques can be used to identify system-level hazards in the ALSA process. Traditionally FHA has been the technique used to identify operational safety risks [1]. More recently, Leveson's STPA has also been used [2]. An FHA output table for the FADEC is shown in Table 1. For our purposes, we consider only the hazards and descriptions for the control thrust function while the aircraft is in motion and do not specify other entries in the table. Table 2 shows the results using STPA.

Table 1: Hazards identified using FHA.

Function	Failure Condition (Hazard Description)	Phase	Effect of Failure	Classifi-cation	Reference to Supporting Material	Verification
Control Thrust	Engine provides no thrust Engine provides too little thrust Engine provides too much thrust Engine is slow to provide commanded thrust (increase or decrease) Engine will not shut down when commanded Engine cannot be controlled – Loss of Engine Thrust Control (LOTC)	Taxi Takeoff Landing Flight				

Table 2: Results of applying STPA.

Accident	System-Level (Operational) Hazards
A-1: Loss of life or serious injury due to aircraft engine A-2: Catastrophic damage to aircraft or other property due to aircraft engine	H0: Ineffective thrust to maintain controlled flight or safe taxi H1: Engine provides no thrust H2: Engine provides too little thrust H3: Engine provides too much thrust H4: Engine is slow to provide thrust (increase or decrease) H5: Engine will not shut down when commanded H6: Complete LOTC

At this point, we identify the top-level safety requirements that prevent hazards or accidents (termed *safety constraints* [2]). The top-level safety requirements for the engine hazards are shown in Table 3. Defining safety requirements in concert with identifying hazards can be more effective, since experienced engineers with the requisite expertise are focused on the specific details of a hazard and immersed in the overall safety context. Often during the hazard-identification activities of ALSA, engineers implicitly assume or identify requirements, identify implicit architecture assumptions or alternatives, or make architecture decisions. For example, engineers may develop a requirement that a combat aircraft will include an ejection set for the pilot. At this point, they can begin to develop an operational model of the top-level system architecture and include an ejection system, along with potential alternative designs.

Table 3: Hazards and safety requirements.

Hazards	Safety Requirements
H1: Engine provides no thrust	SC1: Thrust must be provided at all times when commanded.
H2: Engine provides too little thrust H3: Engine provides too much thrust	SC2: Thrust level must be provided at the commanded level.
H4: Engine is slow to provide commanded thrust	SC3: Engine must provide commanded thrust within the required interval.
H5: Engine will not shut down when commanded	[Relevant safety constraints arising from H5 include SC1, SC2, and SC4.]
H6: Engine cannot be controlled – LOTC	SC4: Engine must respond to all commands. SC4.1: Engine must start when commanded. SC4.2: Engine must shut down when commanded.

Identify Operational Hazards and Contributors

In this step, we detail hazards and identify hazard contributors by incrementally extending the hazard analysis into lower levels of the system operational and architectural hierarchy. Consequently, it is necessary that additional details or working assumptions about the architecture exist and possibly alternative architecture designs for consideration have been defined. These steps incorporate hazard analysis techniques—such as fault tree analysis, event tree analysis, and hazard and operability study from ARP 4761—as well as the STPA and are connected with the identification of safety requirements and the development of a safety architecture.

In the initial activities of this step, we establish the boundaries of the system and its subsystems and define the types of errors that can propagate among them. For the FADEC example, we choose to partition the relevant system into the cockpit (including the pilot), a separate autopilot, and the remainder of the physical aircraft. External elements in the environment may impact the system via sensors or other input (e.g., light entering the aircraft can produce heat within the aircraft). The system-level diagram shown in Figure 3 reflects an architecture in which pilot and autopilot commands to the aircraft's FADEC are separate and parallel.

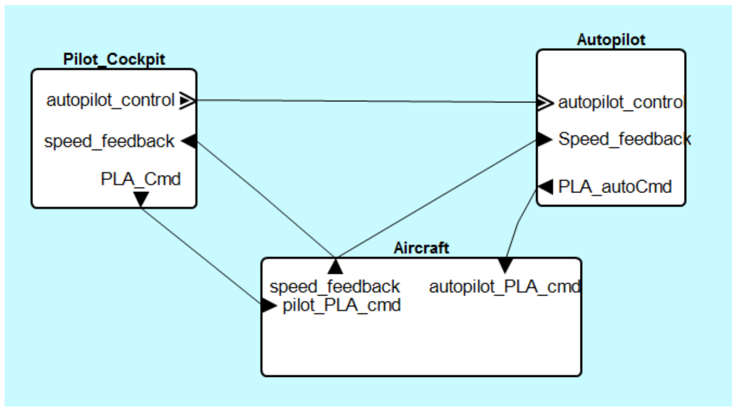


Figure 3: Example aircraft decomposition.

A common way of viewing a system in its operational context is as a control system that involves interactions via Monitored and Controlled Variables. This approach, documented in the *FAA Requirement Engineering Management*

Handbook [14], has its roots with Parnas and Madey [15]. These variables can be used to represent states that characterize unsafe system conditions and interactions. The STPA utilizes a control loop representation for hazard identification [2].

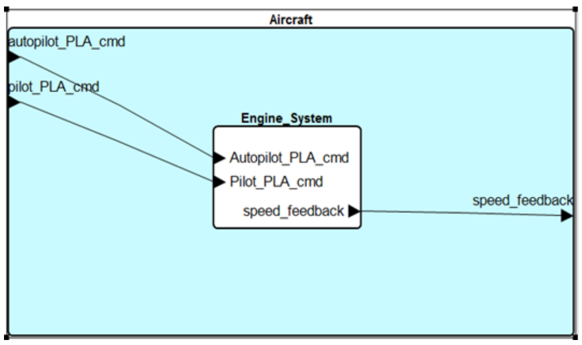
Two principal considerations in identifying hazards are exceptional conditions within architecture elements (characterized using the ALSA error ontology) and mismatched assumptions (mismatched assumption-guarantee contracts between systems) about their interactions. Exceptional conditions and mismatched assumptions are hazardous (undesired) states of a system. Those that pose a threat to the well-being of people or offer the potential for catastrophic consequences to the environment are safety hazards.

As seen in Figure 3, the *Pilot_Cockpit* system provides control commands to the *Autopilot* and the *Aircraft*. We consider the port connections between the elements and choose the error types: no data is sent (service omission), bad data is sent, and data is sent late. We assume that the data is a single content record sent on some schedule. As we define the details of the communication between the components more specifically, we can define the amount of acceptable delay and adjust the model to accommodate these details. This information is summarized in Table 4, whose columns are labeled with the relevant error categories from the error ontology.

Table 4: Interface errors.

Component Interface	Service Errors	Value Errors	Timing Errors	Replication Errors
Pilot_Cockpit to Autopilot	No command to autopilot (may not be a hazard – need details on assumptions of the autopilot system)	Bad value input into Autopilot	Late delivery (since this is specified as a message, potential timing errors need additional analysis)	
Pilot_Cockpit to Aircraft	No command to aircraft	Bad value input into Aircraft	Late delivery	
Autopilot to Aircraft	No command	Bad value	Late delivery	
Aircraft to Pilot_Cockpit	No data	Bad value	Late delivery	Potential for asymmetric missing, value or timing error
Aircraft to Autopilot	No data	Bad value	Late delivery	

The engine system within the aircraft system implementation is shown in Figure 4. For clarity, other internal aircraft components are not included. The FADEC within the engine system can be commanded by either pilot or autopilot input, and the FADEC does a signal selection based on the operational mode. The engine receives a command from the FADEC and provides engine turbine fan speed back to the FADEC.



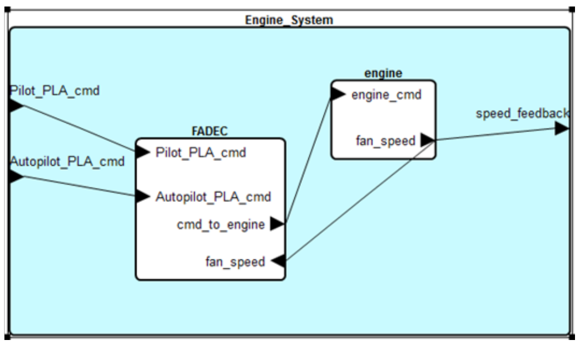


Figure 4: Major engine system components.

Identify Safety Requirements

Operational safety hazards, errors sources, and other contributors to those hazards are used to establish safety requirements—statements about the desired operation and capabilities of a system that address safety hazards. As we have shown, safety requirements arise out of hazards and hazard contributors and can be identified throughout the ALSA process. Consider the example in Table 4; the identification of the hazard that an asymmetric transmission error can occur leads to a requirement to address asymmetric errors in the system.

Develop Safety Architecture Design

Developing a safety architecture is synergistic with the hazard analysis process and the general architecture design efforts. Especially for safety-critical systems, the safety requirements (safety constraints) identified in earlier steps in the process guide the engineering of the system. In safety-guided design, safety requirements drive the overall architecture development and define safety-specific architecture elements, such as redundant hardware, highly reliable communication, and low workload interface designs. Safety requirements significantly influence, and often dictate, architecture and detailed design trade-off decisions and overall system assurance activities.

In ALSA, standard error patterns such as the standard-error state machine shown in Figure 56 are collected in libraries that can be reused across architectures. The safety analysis is tied directly to the architecture by recording the identification of a hazard in the error model for the appropriate AADL component, as shown in Figure 67.

```
error behavior Basic_Three_State
use types ErrorLibrary, FADEC_Error_Library;

events
Bad_Data: error event {Bad_Data} if "occurrences resulting in bad values
being computed";
No_Data: error event {No_Data} if "occurrences resulting in no data
computed";
Repairs: error event if "repairs are made";

states
nominal: initial state; -- component is operating normally
B_Data: state ; -- component is computing and outputting bad values
Failed: state ; -- component is not outputting data

transitions
Data_Bad: nominal -[Bad_Data]-> B_Data;
Major_Fail: nominal -[No_Data]-> Failed;
Fault2: B_Data -[No_Data]-> Failed;
Recovery1: Failed -[Repairs]-> nominal;
Recovery2: B_Data -[Repairs]-> nominal;
end behavior;
```

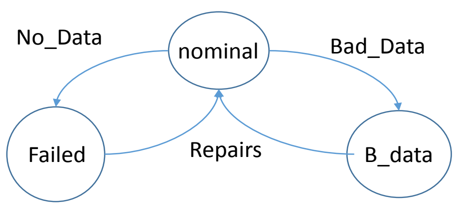


Figure 5: Three-state error model.

```
       EMV2::hazards =>
       (
              [
                     CrossReference => "Hazard H0";
              Description => "Ineffective thrust to maintain controlled flight
or safe taxi";
              Severity => 1;
        ]
       ) applies to Engine_System.ineffective_thrust;
```

Figure 6: Hazard record in an error model.

Errors may be propagated from hardware into software components, or from one component to another, and may arise from the implementation and operation of the component itself, as shown in Figure 78. The error propagations can also lead the system/subsystem hierarchy, as shown in Figure 89. Making these error propagations explicit in the architecture description makes design decisions much more straightforward. This ties the safety architecture into the architecture design activity of the encompassing ACVIP.

```
system FADEC extends Top_Level_Pkg::FADEC

annex EMV2 {**
             use types FADEC_Error_library;
error propagations
autopilot_PLA_Cmd: in propagation {No_Data,Bad_Data,Late_Data};
fan_speed: in propagation
{No_Data,Bad_Data,Late_Data,AsymmetricSpeedFeedback};
pilot_PLA_Cmd: in propagation {No_Data,Bad_Data,Late_Data};
cmd_to_engine: out propagation {No_Data,Bad_Data,Late_Data};
end propagations;
**};
end FADEC;

device engine extends Top_Level_Pkg::engine
      annex EMV2 {**
             use types FADEC_Error_library;
error propagations
engine_cmd: in propagation {No_Data,Bad_Data,Late_Data};
fan_speed: out propagation
{No_Data,Bad_Data,Late_Data,AsymmetricSpeedFeedback};
end propagations;
**};

  end engine;
```

Figure 7: Tying the FADEC to the higher level engine package.

OSATE is extensible using the Extend language. By defining domain specific properties and encoding an algorithm to compute a measure from those properties a new analysis is added to the toolkit. Future work is intended to add to the analysis set and to enhance usability.

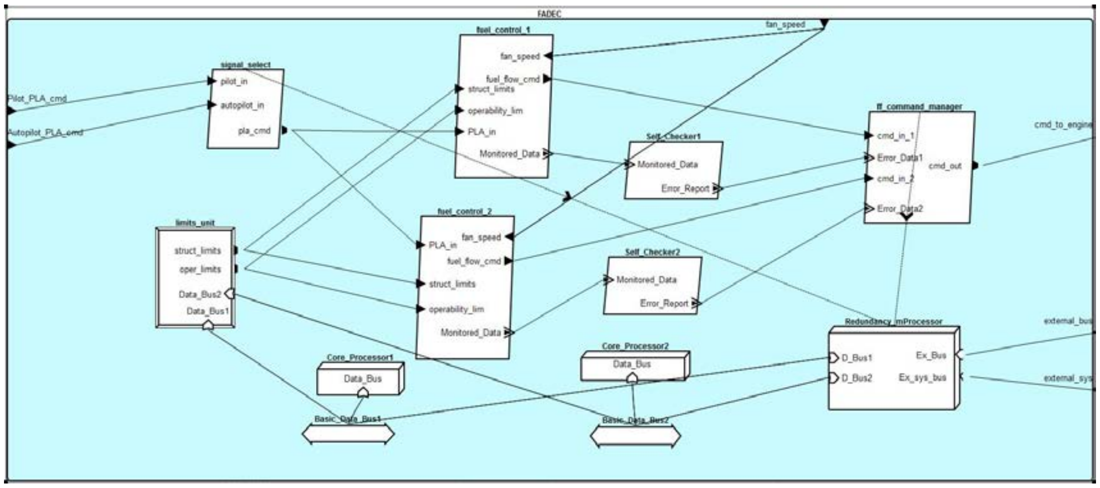


Figure 8: Dual redundant fuel flow.

Conclusion

The ALSA process involves assessing interconnected elements within an architecture, considering the potential EMV2 errors that may apply to the interconnections, and representing each architecture element as an error state machine based on the impacts of these errors. The error model also includes considerations of the operational paradigm and system model of the component. ALSA also involves assessing the interaction paths between architecture components. Within ALSA, systems engineers consider system interaction scenarios in which each component representation is based on an assumed architecture model of the system and assumed operational paradigms (algorithms) that are premised on that model. Each component interaction can be affected by one of the EMV2 error types via its interaction with other components. These are errors output by or received by a component. ALSA is a safety engineering practice that blends existing safety analysis techniques with an error ontology and a model-based architecture representation of the system and its operational context. Using this blend, engineers can more effectively and efficiently discover hazards in the interactions among system components, increase automation to support iterative development, and help to ensure that a safety-critical system is safe.

Acknowledgments

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References

1. SAE ARP4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.
2. Nancy G. Leveson, “A Systems-Theoretic Approach to Safety in Software-Intensive Systems,” *IEEE Trans. Depend. Secure*, 1(1), 2004, pp. 66–86.
3. DeJiu Chen et al., “Modelling Support for Design of Safety-Critical Automotive Embedded Systems.” In *Computer Safety, Reliability, and Security*, Springer, 2008, pp. 72–85.
4. Matthias Biehl et al., “Integrating Safety Analysis into the Model-Based Development Toolchain of Automotive Embedded Systems,” *ACM SIGPLAN Notices*, 45(4), 2010, pp. 125–132.
5. Hans Blom et al., *EAST-ADL: An Architecture Description Language for Automotive Software-Intensive Systems* (White Paper Version 2.1.12), Maenad, 2013.
6. Huafeng Yu et al., “Towards an Architecture-Centric Approach Dedicated to Model-Based Virtual Integration for Embedded Software Systems,” *First Int. Workshop on Architecture Centric Virtual Integration (ACVI), Co-located with MoDELS 2014*, September 2014.
7. Peter H. Feiler and David P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis and Design Language* (SEI Series in Software Engineering), Addison-Wesley, 2012.
8. Peter Feiler et al., *System Architecture Virtual Integration: An Industrial Case Study* (CMU/SEI-2009-TR-017), Software Engineering Institute, Carnegie Mellon University, 2009. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9145
9. Software Engineering Institute, Open Source AADL Tool Environment (OSATE), 2006. https://wiki.sei.cmu.edu/aadl/index.php/Osate_2
10. SAE AS5506B, Architecture Analysis & Design Language (AADL), September 2012.
11. SAE AS5506, Architecture Analysis and Design Language (AADL), Annex E: Error Model Annex, September 2015.
12. Julien Delange et al., *AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment* (CMU/SEI-2014-TR-020). Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=311884
13. Sanjay Garg, *Fundamentals of Aircraft Turbine Engine Control*, NASA, 2011. http://www.grc.nasa.gov/WWW/cdtb/aboutus/Fundamentals_of_Engine_Control.pdf
14. Federal Aviation Administration, *Requirements Engineering Management Handbook* (DOT/FAA/AR-08/32), FAA, 2009.
15. D. Parnas and J. Madey, *Functional Documentation for Computer Systems Engineering* (Version 2, Technical Report CRL 237), McMaster University, Hamilton, Ontario, 1991.

Session 9

Design Space Exploration 2

Thursday 28th, 09:00 – Auditorium St Exupery

Model-compilation challenges
[for Cyber-Physical systems (CPS)]

B. BEN HEDIA, *E.* HAMELIN, *C.* MRAIDHA *&* *S.* TUCCI-PIERGIOVANNI

CEA, LIST, Gif-sur-Yvette, France

{*belgacem.ben-hedia, etienne.hamelin, sara.tucci@cea.fr, chokri.mraidha*}*@cea.fr*

Abstract

There are several "disconnects" which need to be addressed to provide effective means for engineering Cyber Physical Systems (CPS). One of them is how to construct an optimized application starting from high-level specifications taking into account exacerbated interactions with the physical environment and in the same time addressing new paradigms like mixed criticality, and distributed multi/many-cores platforms. We introduce in this position paper a new methodology called **model-compilation** for Cyber-Physical systems. This methodology introduces new concepts and process that can be seen as a specification that tool editors and CPS application developers can integrate (instantiate) in their tool chain or development process.

Keywords: Cyber Physical Systems (CPS), model-compilation, real-time and embedded system, design space exploration, correct-by-design, mix-criticality

1 Introduction

Cyber Physical Systems (CPS) [25] can be considered as the next level of embedded systems [27]. Where an embedded system was mainly concerned with the computational elements for controlling systems, CPS are a network of complex interacting elements which require different physical inputs and outputs and therefore the stress is put on the interaction between separate elements rather than on the computation of a standalone device. However, most of the added value comes from this complex interaction between several devices and the physical environment[26]. The European consumer electronics industry is among one of the strongest players in the fierce global competition for leadership in the embedded systems domain and is now leading the way in CPS.

In CyPhERS project [9] report, for instance, we read: *the principal barrier to developing the field of Cyber-Physical Systems (CPS) is the lack of a theory and of application best practices that comprehend cyber and physical resources in a single unified framework. There are several "disconnects" which need to be addressed to provide effective means for engineering CPS.*

One need that became apparent while building several European project proposals on the development of CPS is to reduce the gap between high-level modeling tools (for instance: Simulink for control and Modelica for physical modeling) aiming at describing functional behaviors and structure, and the actual software implementation on the target platform.

Given the new CPS design challenges [26, 22, 39], the CPS domain needs to move forward from traditional approaches [22], where transition from model to software relies either on fully manual rewriting (a highly error-prone activity), or on code generation tools that can only perform a simple, unoptimized transformation, hard to fine-tune to a given platform architecture. These tools do not address distributed multi/many-cores platforms. Only few research tentatives aim at generating correct-by-design multi-task code from high-level modeling languages, e.g. [40] which addresses the modular generation of multi-rate solvers for Modelica models, and [20] which focuses on reliability aspects.

In our opinion these traditional approaches are not asking the right questions, and limits rapid application prototyping. During the ITEA2 project OPENPROD [19], we have worked on how to construct a software architecture and generate source code starting from a Modelica [3] model for a time-triggered execution platform. At the end of this project, we came to the conclusion that, before starting to build a software architecture and subsequently generate an optimized source code, there are many questions that need to be answered:

- How to generate an optimized software architecture and source code well suited for a given target platform?
- How to integrate the specificity of a given execution platform model (HW platform, communication and runtime support, mix-criticality) during this process, especially with distributed multi/many-cores platforms?
- How to construct the CPS software application structure that will take into account the structural elements available in the high-level model?
- How to construct a CPS process application that respects all requirements described in the high-level model, especially with mixed degree of dependability for different functions and data, through all construction steps, without systematic a posterior V&V methods. In other words how to make the CPS application correct-by-design?

To sum up, we need to construct an approach or methodology that allows the integration of existing tools and techniques and that meets two main concerns:

1. Design space exploration and optimization of software architectures for CPS applications.
2. Correct-by-design methods for CPS applications construction, from high-level model to binary code into execution platform.

This position paper aims at presenting the first rationale of this methodology and at defining key concepts in order to develop an emerging domain of research and technology that we call **model-compilation**: *this term denotes the semi-automated, multi-criteria optimized synthesis of dependable and efficient software implementations distributed on a network of multi/many-cores embedded computers, from abstract system-level models (including multi-physics hybrid control models)*.

Integrated into a model-driven system development methodology, the **model-compilation** methodology will help system, control, software and safety engineers work together to generate a software implementation respecting all their concerns at once, in a holistic, iterative and de-verticalized way. By automating the optimization and trade-offs selection during the software architecture creation, the **model-compilation** methodology will also enable faster development cycles from concept to realization and validation, up to rapid-prototyping of complex, safety-critical applications and hence reduce the development costs and enhance the qualities of cyber physical systems in many industrial domains.

The **model-compilation** methodology can then be seen as a specification that tool editors and developers or CPS application design and development teams can integrate (instantiate) in their toolchain or adopt as a development process with the goal to construct their CPS application. Throughout this paper we illustrate the **model-compilation** methodology concepts on a use-case represented in figure 1.

This position paper is organized as follows: section 2 will introduce the new challenges of software architecture for CPS applications in terms of design space exploration and optimization; section 3 will detail the **model-compilation** methodology and how to integrate its different steps into existing toolchains or how to adopt it as a development process; in section 4 we present how the **model-compilation** methodology answers the challenges expressed in section 2 and its applicability to a use case; section 5 gives an overview of related works in the state of the art; and finally we conclude the paper in section 6

2 CPS challenges

The challenge of implementing a cyber-physical system, as many engineering challenges, can be described in the form of an optimization problem. Some parameters of this optimization problem will be specific to one application domain, however many parameters can be seen as generic problems, only to be weighted differently among use-cases. The optimization problem can be written in the form of:

$$\begin{cases} \text{Min}_{sys}\; Cost(sys) \\ Feasible(sys) \end{cases}$$

In other words: *"find a system implementation (sys) which minimizes the cost function, such that the system is feasible"*. In a traditional mathematical optimization formulation, the $Cost()$ function has as output a single scalar value, whereas the Feasible function is a vector of several feasibility constraints of the form $Feasibile_i(sys) \leqslant 0$

The $Cost()$ and $Feasible()$ functions are specific for each CPS problem; however they are built using generic building blocks, with estimations/weights specific to the problem. Within the Cost function, most CPS domains will enlist the aspects of the system that should be minimized or optimized:

- Cost aspects (accounted positive, to be minimized):
  - Recurring or per-unit costs: e.g. most hardware costs, cabling requirement, devices.
  - Non-recurring costs: e.g. specific HW and SW engineering and development costs, certification costs.
  - System usage costs: e.g. power consumption, maintenance costs.
- Quality and performance aspects (accounted positive, to be maximized):
  - Functional performance of the implemented functionality.
  - Reliability of components.

At this point it is important to note that, since the output of *Cost* function is a scalar evaluation of a proposed system implementation, a weight must be given to all cost components enlisted above: this weight distribution will drive the resolution of necessary trade-offs between performance and cost aspects (e.g. a "low-cost" vs. "premium" strategies). Within the feasibility function many CPS domains will enlist all their domain-specific engineering constraints:

- System minimal viability requirement:

  $Performance \geqslant required$
  (threshold for functionality to be granted)

- System resource constraints:

$$\begin{cases} SW\ memory\ usage \leqslant HW\ memory\ available \\ SW\ proc.\ \&\ comm.\ usage \leqslant HW\ capability \end{cases}$$

- Certification/qualification constraints:

$$Evaluated\ safety\ level \geqslant required\ level$$

The canonical method for minimizing $Cost(sys)$ such that $Feasible(sys)$ is satisfied is to explore the domain of feasible system implementations *sys*, while keeping a track of the best one. Most optimization solvers choose smart strategies to avoid exploring the whole feasible domain, through backtracking and iteration, and split the optimization process into several successive approximations.

The theoretical difficulty is that an accurate cost/performance and feasibility evaluation can only be performed on a concrete system implementation -which makes this optimization-driven engineering method just useless. Moreover, the engineering time spent on finding the optimum of this problem is also part of the non-recurring cost valuation, and may become crucial when time-to-market is a stringent requirement.

It thus becomes necessary to perform "approximate" cost & feasibility evaluations. However an implementation optimal wrt those approximations may consequently not be optimal to the exact cost & feasibility evaluation, should it be performed. It becomes useless looking for an optimal solution, one should rather look for an implementation "good enough" wrt. estimated cost/feasibility. Moreover estimations can be performed at intermediate steps in the design process, i.e. during phases in which the system implementation is only partially known or specified: on system design models. These estimations should however be checked again a posteriori on the concrete system, once implemented. A large effort is dedicated, in the CPS community, to developing tools and design patterns that help design more robust and simple model-based approximations of certain system costs/feasibility evaluations.

As an intermediate conclusion, we assume that CPS systems design is an optimization process, where design constraints and costs are rarely well specified and hard to evaluate accurately, and in which we only look for good-enough, but practical, implementations.

One classical way to speed up approximate optimization problems is to partition variables into mostly-orthogonal sets. An example is usually to split up CPS design into separate dimensions, e.g. with one team exploring the HW trade-offs (against estimated SW requirements), and another team exploring the SW implementation choices. However recent evolution in the CPS industry show that cross-domain optimizations are highly demanded:

- Embedded systems design used to involve both hardware and software engineers to explore cross-domain trade-offs for higher performance control and communication systems,
- Mechatronics systems design used to mix previous embedded systems teams with mechanical engineers to explore tighter system integration,
- Now Cyber-Physical Systems implies to cooperate with multi-physics modelling, instrumentation and control engineers, for flexible, distributed systems interacting with humans and physical systems.

The whole **model-compilation** challenge relies in:

- identifying how much of the approximate-optimization problem solving can be assisted by automatic computer reasoning;
- defining cross-domain interfaces that allow control, software, and hardware engineers cooperate on coherent models, to allow for multi-domain trade-off selection;
- setting-up practical tools through which multi-domain experts can cooperate with computer-aided design choices, in order to develop feasible, good-enough system implementations.

We call this process **model-compilation**, as a tribute to the software compilation process, in which a high(er)-level language is automatically transformed into a lower-level (assembly, then binary) implementation, through a roughly-optimizing compilation process.

When compilers were introduced, an assembly expert could certainly write a fine-tuned, high-performance implementation of a given algorithm, it would take so much time that the system performance drawback of using a compiled language is undoubtedly outweighed by the productivity of the software engineer. Today assembly is used only in very specific situations, and the performance drawback is moreover significantly reduced thanks to compiler and high-level languages improvements. We expect similarly that model-driven engineering will take over, thanks to general adoption of both model-driven engineering and step-wise improvements in the implementations generated by model-compilers, so that within a couple decades most of a CPS's implementation will be semi-automatically generated, and CPS software engineers only have to implement specific aspects such as low-level glue between HW and the generated SW part.

The **model-compilation** process can be seen as a multi-domain toolbox that enables control engineers, embedded software architects, and hardware designers share their domain-specific requirements and iterate on multi-domain trade-offs. The control engineer in particular expects to:

- express the CPSystem's functionality, often in the form of a block chain involving sensing, filtering and prediction, communication and actuation;
- validate system functionality, stability, robustness and reactivity in various scenarios, usually through MiL (model-in-the-loop) simulations involving interconnecting the controller model with a model of the physical plant under control;
- assess, at each iteration of the domain-space exploration process, that a given software/hardware implementation is functionally correct w.r.t. the original control model, e.g. through semantically-correct-by-construction software generation, or MiL or SiL (Software-in-the-Loop) validation.

The hardware engineer expects to:

- define a hardware architecture made of ECUs interconnected through networks, each node having specific computing structure (e.g. a specific SoC with single/multi/many-core architecture), and specific limitations (e.g. memory/computing/bandwidth resource);
- perform preliminary safety assessment of the effects of common HW failure modes:
- negotiate HW/SW trade-offs with SW engineer, e.g. upgrade a CPU computing power, or downgrade SW functionality.

The software architect expects to:

- assess software architectures, at several abstraction levels (functional component, task, runnable, or source-code block), for feasibility, performance, and safety metrics;
- explore software architectures with computer assistance towards a nearly-optimal choice;
- automatically generate a correct source code implementation of the chosen software architecture, that suits a specific RTOS API, and if necessary re-write only the performance-critical part of it;
- iterate quickly with hardware or control engineers on SW/control or SW/HW trade-offs.

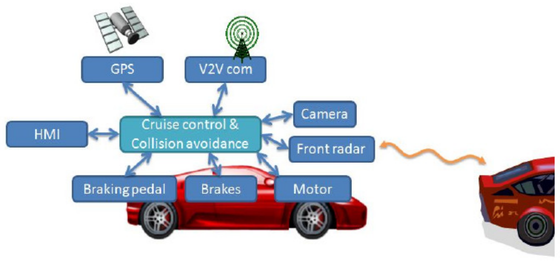


Fig. 1: Adaptive cruise control and collision avoidance system

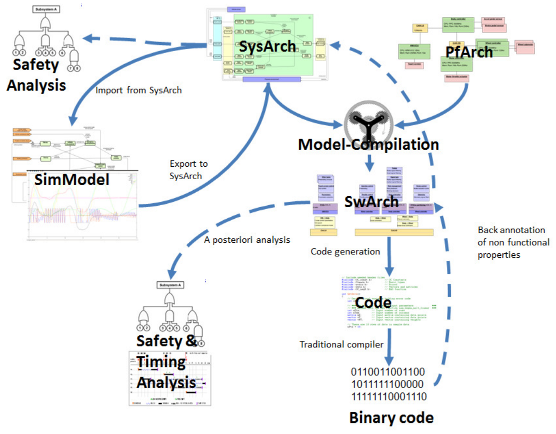


Fig. 2: Global **model-compilation** methodology

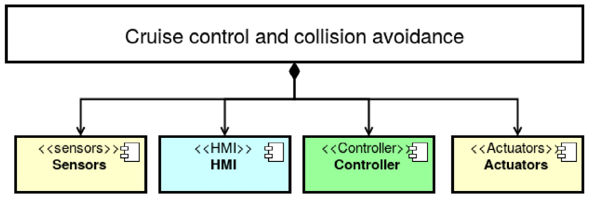

Cruise control and collision avoidance
<<sensors>> Sensors
<<HMI>> HMI
<<Controller>> Controller
<<Actuators>> Actuators

Fig. 3: Conceptual systems architecture

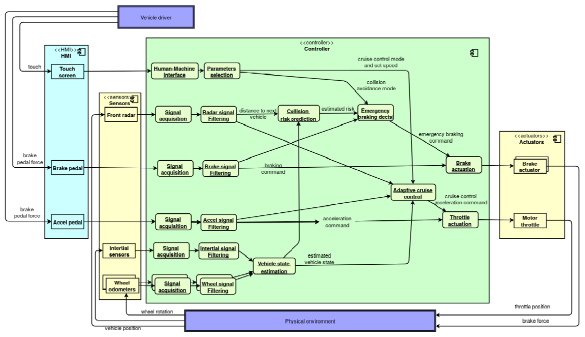


Fig. 4: SysArch: **Sys**tems **Arch**itecture model

3 Model-compilation (methodology & approach)

In order to best present the complete end-to-end approach, and how it is integrated within a model-driven development process, this section highlights the application of the **model-compilation** methodology on an adaptive cruise control and collision avoidance system. This system is sketched by figure 1.

Figure 2 gives an overview of the **model-compilation** process. Like a third generation programming language compiler, the model-compiler is composed by a front-end, middle-end and back-end parts.

3.1 Front-end: from multiple heterogeneous high-level models

The system is first designed as a conceptual systems architecture, using a general purpose or domain specific architecture modelling tool, where main system constituents and their relations are identified as illustrated on the following Unified Modelling Language (UML) diagram. The Cruise control and collision avoidance system is composed of a set of sensors, actuators as well as a human-machine interface (HMI) and a controller.

At this level, a first safety analysis and risk assessment can be conducted to estimate the level of risk associated with specific items (elements of the architecture). Since we are considering an automotive use case, for risk assessment we will use the concepts coming from the functional safety standard for automotive equipment, the ISO26262 standard. Accordingly to ISO26262, the risk level is specified through an Automotive Safety Integrity Level (ASIL). A Safety Goal is then defined for each hazardous event identified (e.g. “when the brake pedal is pushed, a braking torque must be applied to the wheels”) as well as a set of essential safety requirements.

SysArch: systems architecture model The component-based design is then detailed as an abstract data flow diagram, illustrated below. This diagram is usually hierarchic, for sake of representation and understanding. This diagram will later be referred to as the functional architecture, or systems architecture: in brief SysArch.

This SysArch can efficiently be used to perform several design activities relevant at the systems abstraction level, as required by the ISO26262 standard. This activity includes the definition of functional safety requirements and their allocation to this preliminary functional architecture. At this phase the System Hazard Analysis (SHA) is conducted to study the propagation of failures across the system architecture. A preliminary safety assessment is conducted as well through Fault Tree generation and qualitative Analysis (FTA), Failure Mode Effects Analysis (FMEA) and Common Cause Analysis (CCA). These methods aim at analyzing fault propagation through the system and help in the definition of safety goals and ASIL criticality level. For instance, a functional chain that supports a ASIL C safety goal should rely on functional blocks assigned a criticality of ASIL C or higher.

This functional system architecture model can also be used for the specification of timing requirements. Some chains of functions can be identified and used to define global end-to-end deadlines (e.g. “latency from brake pedal

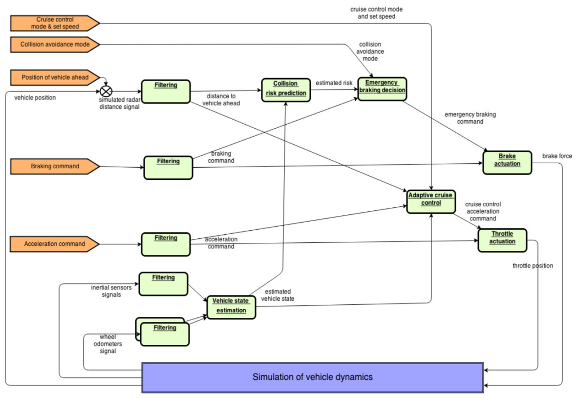


Fig. 5: SimModel: **Sim**ulation **Model**

signal acquisition to the corresponding brakes actuation should always be under 10ms").

SimModel: the Simulation Model Then, for the control engineering team to work out the regulation principles, the data flow model is turned into a simulation model that will be used in particular to design and validate the controller regulation parameters, and assess their stability and robustness. This simulation is built from the same architecture, except that it includes a simulation of the environment (e.g. of the vehicle dynamics). The Simulation model is specified in a multi-physics modelling language.

Some blocks of the original systems diagram have been abstracted away: mainly the controller's interfaces to the external world are replaced with simulation stubs. For instance, some sensor blocks have been replaced by input signals (the orange blocks) that connect to external signal scenarios (e.g. a list of cruise control set speeds, brake pedal positions, etc. at given simulation time instants). Some other blocks (like actuators and sensors) are replaced with transparent connections to signals from/to the vehicle dynamics simulation or a simulation of the sensor or actuator's internals. The "human-machine interface" and "parameter selection" blocks are also abstracted away by the "cruise control mode & set speed" input signal block, because these blocks represent a graphical user-interface and menu handling functions, therefore are not suitable for multi-physics simulation.

Although the final software implementation of the whole cruise control and collision avoidance system will be realized by purely discrete computations, at this point both the vehicle dynamics and the controller may be specified with either discrete or continuous (i.e. time-differential) equations. Typically the "vehicle state estimation" block in the controller will integrate differential equations. Many parameters evaluated during this simulation will be used to drive the **model-compilation**.

PfArch: the Platform Architecture In parallel to the simulation modelling activity, a hardware platform model, here named PfArch, is defined. It can be represented by a graph of interconnected computing, communication, sensing/actuating resources as illustrated below. Each computation resource should be annotated with its own specific resource constraints, e.g. CPU speed, RAM and ROM memory limits, hardware peripherals available, maximum criticality allowed (i.e. ASIL qualification level); and similarly buses may be annotated with a bandwidth or any other required by later feasibility analyses.

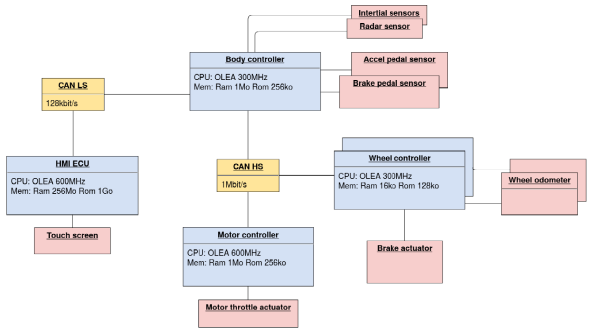


Fig. 6: PfArch: **P**lat**f**orm **Arch**itecture

In many cases, it can be allowed to change this architecture to some extent in order to meet the needs of the final software implementation. In this case, some production rules may be defined, together with an associated cost value, that allow the computer-assisted optimization to create a new platform architecture from the initial one if the application does not meet the constraints of the initial platform architecture (e.g. "another Electronics Component Unit (ECU) can be added on a bus, at some cost; a bus can be replaced with one with higher bandwidth, at some cost, etc.").

3.2 Middle-end: model-compilation into software architecture

The middle-end part is at the heart of the **model-compilation** approach and lies in the optimized and efficient synthesis, from a system architecture model (SysArch), of a software architecture (SwArch) mapped onto the platform architecture (PfArch), that simultaneously satisfies safety, timing, resources and behaviour requirements. The **model-compilation** is implemented by a combination of optimization algorithms and heuristics for transforming, in an optimized manner, a system-level model into a software implementation mapped onto a platform architecture model:

1. Optimisation algorithms aim at finding a solution correct-by-construction, i.e. the transformation itself is proven to guarantee that the SwArch produced respects all the constraints related to timing, safety, behaviour, resources, etc. Such approach called multi-staged optimization approach for multi-objective optimization has been already investigated in our previous works [30, 42]. The approach was able to find correct solutions which respect a wide range of constraints including causality and schedulability, while identifying design trade-offs. Note that producing correct solutions means that the analysis supposed to verify correctness is included in the optimization algorithm as a set of constraints.
2. Heuristics fall in a trial-and-error paradigm: here the SwArch obtained with the heuristics must be anal-

ysed a-posteriori (i) to establish if all the constraints are respected and (ii) if the required trade-offs are met. Heuristics will be applied whenever the overall complexity of the problem hinders the application of deterministic optimisation algorithms. One example of efficient trial-and-error strategy is to use coarse estimates of parameters to produce several nearly-optimal feasible solutions (e.g. use only a coarse estimate of CPU utilization factor for task creation step), then apply fine-grain analysis only to those few solutions (e.g. validate schedulability with a detailed period & WCET analysis).

For the automated transformation to be possible, either as correct-by-construction generation or as analysis-driven trial-and-error, all concerns must be first formalized and quantified. Some requirements will be quantified as quantitative relationships that must be verified, whereas some other requirements will be defined as the maximization of some quantified objective. To enable multi-concern trade-offs, a single objective (usually a weighted sum of all sub-objectives) shall be defined. Different weights could be selected depending on the application. [13, 14] aimed to integrate the specificity of an execution paradigm (RTOS) during the **model-compilation** process by integrating some characteristics of the execution platform when the SwArch is generated. Since then, the generation process became more accurate, since it now integrates information about the paradigm of the execution platform.

It should be noted that both optimization algorithms and heuristics accept some degree of parameter uncertainty. For instance, worst case execution time estimates could be extracted from in-situ measures (when reusing a function or task), or safe over-approximations (when using static analysis tools on the actual software), or only coarse estimates when the function has not yet been implemented/generated.

SwArch: Software Architecture intermediate representation Concretely at the end of the **model-compilation** process, all functional blocks from the SysArch model are transformed into some software code mapped onto one or several tasks or runnables, each allocated to a CPU, and all data or events (the edges of the SysArch diagram) are mapped to variables or inter-task communication messages or to bus messages; all tasks and signals have scheduling parameters assigned, in such a way that on each computation unit all tasks are schedulable, messages are schedulable on each bus, all end-to-end latency constraints are met, no CPU resource is exhausted, items of different criticality levels are separated and monitored by safety elements qualified at the appropriate criticality level, etc. Several reports may be produced together with the SwArch, for instance: a schedulability report for each CPU, an analysis of end-to-end latency constraints, a safety analysis at the software architecture level demonstrating the respect of criticality-related constraints, etc.

In the following illustration, functions of different criticality levels are mapped onto tasks. Tasks of different ASIL levels are separated either physically or in software by a SEooC RTOS providing sufficient partitioning mechanisms. A first work addressing the co-simulation of the SimModel

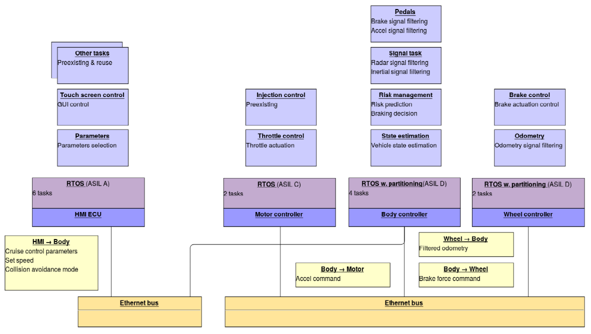


Fig. 7: SwArch: **S**oft**w**are **Arch**itecture output

and SwArch is provided in [32].

3.3 Back-end: transformation into concrete target platforms

A rather simple model-to-code transformation back-end translates the SwArch model, into a set of files that actually implement in a software language e.g. C, all these tasks, and properly configure each CPU and RTOS instance. A first work [13, 14] was carried out to integrate a generic services of an execution paradigm (RTOS) during a **model-compilation** process and integrate some information about execution plate-form in the generated SwArch, with the intention of making easier the Back-end stage. The back-end of the **model-compilation** process then relies on off-the-shelf C compilers to generate the binary file that will be loaded on each ECU. This translation back-end is specific to the RTOS since it relies on the specific RTOS application programming interface (API) and services for the use of temporal control flow and inter-task messaging for instance. For some blocks of the SysArch where only a wrapper/placeholder was given in the simulation model, an empty task or runnable is generated together with the inter-task communication infrastructure, so that it still remains easy to manually add the manual implementation part. The wrapper & placeholder method facilitates the integration of existing tasks into the **model-compilation** process.

3.4 Design iterations

For each artifact of the SwArch model (task, runnable, message, etc.), a traceability annotation ensures that a straightforward link identifies the SysArch functional block or signal it was generated from and the design constraints that apply.

When the generated code is run on the actual platform properly instrumented, it is possible to measure relevant metrics at actual runtime: for instance measure actual execution times with real-time tracing/debugging tools. These runtime metrics can then, thanks to the traceability feature, be retro-annotated onto the SwArch model, then on the SysArch model. In many cases these actual metrics are much more accurate than the estimations given at first (if any), and can be used to perform another run of the **model-compilation** for a more optimized or more accurate synthesis. This iterative design approach also allows an expert engineer to identify on actual execution abnormalities, violations of unexpressed or soft requirements, bottlenecks, sources of possible optimizations unseen before, etc. In this case, the engineer can annotate the SysArch model with different constraints that drive some steps of the **model-compilation** towards his preferred solution, applying the **model-compilation** process again.

4 Model-compilation methodology assessment

Here we address the question whether, and how far, the challenges defined in sections 1 and 2 can be resolved by implementing the approach illustrated in section 3.

Applicability of model-compilation approach The **model-compilation** methodology would be applied by first implementing all (semi-)automated analysis and optimization algorithms into already-existing tools, and then applying the development method to realistic use-cases.

1. **Integration/implementation by existing tools.** To effectively support the model-compilation process, analysis/optimization plug-ins and model transformations must be integrated to existing model-based development (MBD) tools:

    - A set of model analysis/optimization techniques will refine different aspects of the SwArch model. Most model-based development (MBD) tools allow to define specific languages and therefore support the SwArch models; moreover they support plug-ins and extensions mechanisms, within which feasibility, performance evaluation and optimization techniques can be implemented. Our previous work [13, 19, 14, 42, 30, 33] address different analysis & optimizations techniques as a proof of concept of this methodology.

    - The development process defined here certainly needs to rely on several engineering tools, therefore some bridges are required to support a seamless workflow along several iterations of individual steps of the **model-compilation** methodology. This generic toolbox can then be tailored by each engineering team into a product-specific development method, in which eventually a set of process guidelines can also be enforced through tool usage rules. Each process step should moreover be formalized with a set of hypotheses and guarantees, so that it could be eventually possible to verify the overall coherency of each specific workflow – this point will be further detailed in forthcoming publications.

2. **Realistic use-case application.** The illustrative use-case detailed in section 3 proves that the approach is applicable, through several analysis & transformation steps, on a virtual use-case representative of a realistic, non-trivial, industrial system.

Productivity enhancements The CPS development productivity will be illustrated on three main axes:

1. **(semi-)automation of individual steps.** Several tasks are automated or computer-aided, especially those most labor-intensive, with low systems-engineering added value, or most error-prone. Common MBD tools usually generate hardly-readable source-code in only one iteration, are poor at engineer interaction, with limited (if any) analysis or user-guided optimization facilities.

2. **Overall workflow acceleration.** With **model-compilation**, the CPS development workflow enables an exploration and optimization of SwArch. This allows to quickly evaluate several configurations of the software before choosing which one to apply for the final source-code generation. Existing tools generally generate code that is poorly structured, therefore adaptation to a specific execution platform (RTOS) or HW platform is very costly. Architectural exploration is subsequently only manual, and very long. Within the **model-compilation** methodology, all necessary information for the SwArch exploration phase are integrated, so that exploration steps are quickly iterated.

3. **Early feasibility assessment.** The feasibility and cost/performance evaluations performed at different phases during architecture exploration allow to early focus only on feasible alternatives.

5 Related works

This section presents the current state-of-art of some research areas relevant for CPS development, which in our opinion need further consideration.

Model-based methodologies for safety and timing Considering the whole body of work on development methodologies for systems and software is of course unfeasible. Nevertheless we are interested in giving an overview particularly focused on safety, timing and methodological management based on the use of modelling languages, which narrows the scope of this literary review.

Methodologies for safety management have been mainly proposed by T.Kelly and his group [18, 38, 17, 16]. These methodologies focus on how to build a safety case both at system and software level. Safety case patterns are proposed to ease this task [16], however, these approaches propose only an abstract methodology, i.e. they are not focusing on the use of either particular language (e.g. SysML, UML, etc) or models. This lack of concretization also implies a lack of methodology automation. A model-based approach has been proposed in the FP7 MAENAD project, where the GMP pattern methodology developed in TIMMO and TIMMO-2-USE projects has been instantiated on EAST-ADL models. This concretization allowed the automation of some safety activities such as FMEA and static and temporal FTA; still the automation support is partial with respect to the whole safety management, and safety case patterns were not integrated.

On the timing side, while a vast number of model-based approaches have been proposed for performance prediction [29], methodologies enabling analysis of timing constraints are more recent and are gaining a growing interest with the increasing complexity of embedded real-time systems [8, 4]. A UML based methodology based on the Y-chart principle has been proposed18 which then envisages the use of mapping algorithms (**model-compilation**) but not specifically focusing on timing analysis. A UML-based methodology for the analysis of objected oriented models but without proposing an automated mapping step has also been proposed [21]. Mraidha et al. [33] proposed a MARTE based methodology based on a scenario-based mapping and enabling schedulability analysis of mono-processor systems.

Note that the only methodological attempt, we are aware of, about the integration of safety and timing has been pursued in TIMMO and TIMMO-2-USE projects with the GMP pattern methodology. However this methodology is very general and do not specify how safety and timing management interact during the development process, this specification is left to the system engineer. New methodologies proposing a tighter integration of safety and timing activities throughout the development processes are needed to better support the development of CPS with mixed-critical constraints.

Model-compilation In the development of cyber-physical systems, abstraction levels must be used to manage complexity [24]. Industrial standards (like the automotive AUTOSAR [1] and the Model-Driven Architecture from the OMG [24]) and academic frameworks (including the Platform-Based Design [37]) recommend system development along the lines of the Y-chart approach [23]: a functional model representing the system functions and the signals exchanged among them (model often represented by Simulink-like models) is deployed onto an execution platform model consisting of nodes, buses, tasks and messages. Throughout this paper we call **model-compilation** the refinement of a functional model in its "execution counterpart": functions concretised by software modules (code) deployed across sofware (middleware, OS, etc...) and hardware (CPUs, memories, buffers, etc..) execution architectures, mostly distributed (as for CPS). Correctness of **model-compilation** implies that this functional execution counterpart must respect timing and safety constraint when running in the target platform.

On this topic, the current state of the art has been developed by two rather separated communities: the formal methods community focusing on functional model scheduling [28, 7, 36, 34] (with implicit assumption about a deployment on uni-processor platforms) and the real-time community focusing on allocation and scheduling of tasks models (with a large use of optimization and design exploration techniques), leaving the function-to-software transformation to the designer [15, 35, 11, 31, 10].

Providing a holistic optimization approach for **model-compilation**, i.e. optimization applied to the deployment of a functional model over a multi-processors architecture (multi-core and/or distributed). In this direction recent works, such as Mehiaoui et al. [30] and Wozniak et al.56 proposed a two-staged multi-objective optimisation technique able to output schedulable solutions for functions to be deployed in a distributed architecture. These approaches must be extended to specifically address safety constraints and mixed-criticality. Another important research topic about **model-compilation** concerns model transformations for Matlab Simulink models. While semantics preserving translations of Simulink models into tasks exist for discrete models, the translation of continuous blocks is an open research issue.

Physical modelling Historically, physical modelling and discrete control have mostly been treated as separate engineering activities. However, over the years, due to an increasing demand in tools covering a wider range of product lifecycle, the need for more integrated approaches has emerged [41].

Today, industrial applications of physical modelling are handled by tools providing informal semantic models mostly inspired from results in the field of continuous system simulation. Discrete aspects, despite being essential even in pure physical applications, are generally poorly supported by modelling tools, sometimes leading to important issues as soon as discrete aspects have to tightly interact with continuous ones [12, 5, 6].

Currently, tools essentially resort to their own semantic models, believed to be compatible with the operational requirements of some emerging standards such as Modelica [3] and FMI [2]. "High-level" proposals such as Modelica, VHDL-AMS and Verilog-AMS, and "low-level" proposals such as FMI attempt to provide a common basis for different users using different tools to describe models in such a way that tool interoperability and model exchange are hopefully possible. However, to the best of our knowledge, none of these proposals currently attempt to fully formalize the operational semantics required to reach the desired portability of hybrid models, especially their continuous-time part [6]. This means that a sound operational semantics for physical models is yet to be defined so that models coming from various sources (so including physical models) have to be combined in a sound way.

On the other hand, the situation is more comfortable in pure discrete-time applications since many semantically sound approaches have been proposed over the years to cope with them, one of the most prominent among them being the synchronous approach. Many tools targeting embedded applications (among which CEA's tools) support a form of synchronous approach. This approach allows one to reason about logical timing properties of models at a high level of abstraction, even before target code has actually been generated. This makes the synchronous level of abstraction an attractive "common denominator" for models that need to be checked for semantic compatibility and for qualification for embedded purposes. However, no tool allows physical models to be rigorously transformed into this intermediate form currently [6], which clearly constitutes an important challenge to be addressed.

6 Conclusion

In this position paper, we have defined a roadmap for a method of development for cyber-physical systems, to address the new challenges of rapidly designing multi-viewpoint optimized implementations, that we call model-compilation. The proposed approach addresses two main concerns: design-space exploration and optimization of a software architecture, and correct-by-design construction from high-level models to source code and binary for a given execution platform. The model-compilation methodology can then be seen as a general specification that tool editors and CPS design teams can integrate (instantiate) in their toolchain, or adopt as a development process.

This roadmap is deemed feasible through several previous work and publications by the authors, but many aspects will be further detailed to confirm that the whole model-compilation approach can be efficiently implemented in industrial practice.

References

[1] Autosar 4.0 specifications. `http://www.autosar.org/`.

[2] Fmi standars. `http://www.fmi-standard.org`.

[3] Modelica. `http://www.modelica.org`.

[4] Cesare Bartolini, Antonia Bertolino, Guglielmo De Angelis, and Giuseppe Lipari. A uml profile and a methodology for real-time systems design. In *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on*, pages 108–117. IEEE, 2006.

[5] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. *Journal of Computer and System Sciences*, 78(3):877–910, 2012.

[6] Simon Bliudze and Sébastien Furic. An operational semantics for hybrid systems involving behavioral abstraction. In *Proceedings of the 10th International ModelicaConference*, number EPFL-CONF-199085, pages 693–706. Linköping University Electronic Press, Linköpings universitet, 2014.

[7] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, Stavros Tripakis, and Peter Niebert. From simulink to scade/lustre to tta: a layered approach for distributed embedded applications. In *ACM Sigplan Notices*, volume 38, pages 153–162. ACM, 2003.

[8] Rong Chen, Marco Sgroi, Luciano Lavagno, Grant Martin, Alberto Sangiovanni-Vincentelli, and Jan Rabaey. Uml and platform-based design. In *UML for Real*, pages 107–126. Springer, 2003.

[9] CyPhERS. Cyber-physical european roadmap & strategy, 2014.

[10] Werner Damm, Alexander Metzner, Friedrich Eisenbrand, Gennady Shmonin, Reinhard Wilhelm, and Sebastian Winkel. Mapping task-graphs on distributed ecu networks: efficient algorithms for feasibility and optimality. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 87–90. IEEE, 2006.

[11] Cagkan Erbas. *System-level modelling and design space exploration for multiprocessor embedded system-on-chip architectures*, volume 132. Amsterdam University Press, 2007.

[12] Sébastien Furic and LMS Imagine. Enforcing reliability of discrete-time models in modelica. In *Proceedings of the 8th International Modelica Conference*, 2011.

[13] Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, Saddek Bensalem, and Jacques Combaz. Towards time-triggered component-based system models. In *The Tenth International Conference on Software Engineering Advances (ICSEA)*, 2015.

[14] Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, and Saddek Bensalem. Externalisation of time-triggered communication system in bip high level models. *JRWRTC 2014*, page 41, 2014.

[15] Arne Hamann, Razvan Racu, and Rolf Ernst. Formal methods for automotive platform analysis and optimization. In *In Proc. Future Trends in Automotive Electronics and Tool Integration Workshop (DATE Conference), Munich*. Citeseer, 2006.

[16] Richard Hawkins, Kester Clegg, Rob Alexander, and Tim Kelly. Using a software safety argument pattern catalogue: Two case studies. In *Computer Safety, Reliability, and Security*, pages 185–198. Springer, 2011.

[17] Richard Hawkins, Ibrahim Habli, and Tim Kelly. Principled construction of software safety cases. In *SAFECOMP 2013-Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, page NA, 2013.

[18] Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. A new approach to creating clear safety arguments. In *Advances in systems safety*, pages 3–23. Springer London, 2011.

[19] B. Ben Hedia and E. Hamelin. Model to embedded real-time real time embedded code transformation, 2012.

[20] Jia Huang, Simon Barner, Andreas Raabe, Christian Buckl, and Alois Knoll. A framework for reliability-aware embedded system design on multiprocessor platforms. *Microprocessors and Microsystems*, 38(6):539 – 551, 2014.

[21] Dongxi Jin and David C Levy. An approach to schedulability analysis of uml-based real-time systems design. In *Proceedings of the 3rd international workshop on Software and performance*, pages 243–250. ACM, 2002.

[22] S.K. Khaitan and J.D. McCalley. Design techniques and applications of cyberphysical systems: A survey. *Systems Journal, IEEE*, 9(2):350–365, June 2015.

[23] Bart Kienhuis, Ed F Deprettere, Pieter Van Der Wolf, and Kees Vissers. A methodology to design programmable embedded systems. In *Embedded processor design challenges*, pages 18–37. Springer, 2002.

[24] Stefan Kugele, Wolfgang Haberl, Michael Tautschnig, and Martin Wechs. Optimizing automatic deployment using non-functional requirement annotations. In *Leveraging Applications of Formal Methods, Verification and Validation*, pages 400–414. Springer, 2008.

[25] Edward A. Lee. Cyber-physical systems - are computing foundations adequate? In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, October 2006.

[26] Edward A. Lee. Cyber physical systems: Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[27] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Lee and Seshia, 1 edition, 2010.

[28] Roberto Lublinerman and Stavros Tripakis. Modular code generation from triggered and timed block diagrams. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 147–158. IEEE, 2008.

[29] Jack E Matson, Bruce E Barrett, and Joseph M Mellichamp. Software development cost estimation using function points. *Software Engineering, IEEE Transactions on*, 20(4):275–287, 1994.

[30] Asma Mehiaoui, Ernest Wozniak, Sara Tucci-Piergiovanni, Chokri Mraidha, Marco Di Natale, Haibo Zeng, Jean-Philippe Babau, Laurent Lemarchand, and Sébastien Gerard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. *ACM SIGPLAN Notices*, 48(5):121–132, 2013.

[31] Alexander Metzner and Christian Herde. Rtsat–an optimal and efficient approach to the task allocation problem in distributed architectures. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 147–158. IEEE, 2006.

[32] Matteo Morelli, Yasmina Seddik, Marco Di Natale, Chokri Mraidha, and Sara Tucci-Piergiovanni. Simulation-driven optimization of real-time control tasks. In *Proceedings of International Conference on Embedded Software and Systems (ICESS)*, 2015.

[33] Chokri Mraidha, Sara Tucci-Piergiovanni, and Sebastien Gerard. Optimum: a marte-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.

[34] Thomas M Parks, Edward Lee, et al. Non-preemptive real-time scheduling of dataflow systems. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 5, pages 3235–3238. IEEE, 1995.

[35] Traian Pop, Paul Pop, Petru Eles, and Zebo Peng. Analysis and optimisation of hierarchically scheduled multiprocessor embedded systems. *International Journal of Parallel Programming*, 36(1):37–67, 2008.

[36] Marc Pouzet and Pascal Raymond. Modular static scheduling of synchronous data-flow networks. *Design Automation for Embedded Systems*, 14(3):165–192, 2010.

[37] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, (6):23–33, 2001.

[38] Mark A Sujan, Floor Koornneef, Nick Chozos, Simone Pozzi, and Tim Kelly. Safety cases for medical devices and health information technology: Involving healthcare organisations in the assurance of safety. *Health informatics journal*, 19(3):165–182, 2013.

[39] Paulo Tabuada. Cyber-physical systems: Position paper. In *NSF Workshop on Cyber-Physical Systems*, 2006.

[40] Bernhard Thiele, Martin Otter, and Sven Erik Mattsson. Modular Multi-Rate and Multi-Method Real-Time Simulation. In Hubertus Tummescheit and Karl-Erik Årzén, editors, 10$^{\text{th}}$ *Int. Modelica Conference*, Lund, Sweden, May 2014.

[41] Vincent Berthoux Sébastien Furic Loïc Wagner and LMS Imagine SA. Statecharts as a means to control plant models in lms imagine. lab amesim.

[42] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci Piergiovanni, and Sebastien Gerard. An optimization approach for the synthesis of AUTOSAR architectures. In Carla Seatzu, editor, *Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA 2013, Cagliari, Italy, September 10-13, 2013*, pages 1–10. IEEE, 2013.

Pareto-efficient deployment synthesis for safety-critical applications in seamless model-based development

Sergey Zverlov fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
Email: zverlov@fortiss.org
Maged Khalil fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
Email: khalil@fortiss.org
Mayank Chaudhary fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
Email: chaudhary@fortiss.org

Abstract—**Increasingly complex functionality in automotive applications demand more and more computing power. As room for computing units in modern vehicles dwindles, centralized architectures - with larger, more powerful processing units - are the trend. With this trend, applications no longer run on dedicated hardware, but share the same computing resources with others on the centralized platform. Ascertaining efficient deployment and scheduling for co-located applications is complicated by the extra constrains which arise if some of them have a safety-critical functionality.**

Building on our pre-existing design space exploration solution, we integrated safety constraints, such as ASIL and HW failure rates, as well as practical aspects, such as component costs, and extended the approach to allow for multi-criteria optimization. The work was implemented into our seamless model-based research CASE tool AutoFOCUS3 and evaluated using a non-trivial industrial-inspired case study. The solution is capable of synthesizing deployments together with corresponding schedules, which satisfy different safety and resource constraints. The deployments can subsequently be integrated into the safety case argumentation of AutoFOCUS3, leveraging the tool's seamless capabilities to support safety evidence and certification.

Moreover, we are not interested in merely valid solutions, but in good ones. We hence developed a multi-objective optimization algorithm, which synthesizes solutions pareto-optimized for safety, resource usage, timing and any other constraints the user defines. Our approach demonstrates the feasibility and effectiveness of using formal methods to generate correct solutions for safety-critical applications, increasing the confidence and validity of safety cases.

Index Terms—**Design Space Exploration, AutoFOCUS, Optimization, SMT, Safety, Resource Optimization, Deployment, Scheduling**

I. Introduction

Connected vehicles and advanced driver assistance systems (ADAS) are just a few of the latest technological drivers increasing the demand on computing power in classical domains of embedded systems development, such as the automotive sector. Given the already high number of ECUs (electronic control unit) in today's automobiles, there is a clear trend towards centralized architectures featuring larger, more powerful (potentially multi-core) platforms. This trend lends increased urgency to the problem of mapping logical computational tasks to the hardware (HW) nodes that will execute them. This task is intrinsically difficult in embedded systems, because their limited resources add many constraints to the deployment problem that have to be respected, e.g., execution timing, memory constraints, power consumption, bus loads and many more. If the applications to be deployed are safety-relevant, this adds a new dimension to the deployment problem, increasing the difficulty many-fold. Safety standards dictate design patterns, such as partitioning according to criticality, to maintain freedom from interference, but also set requirements for the hardware nodes' reliability. This adds multiple criteria (some orthogonal) to both the problem space (software components) and the solution space (hardware nodes). Mapping logical/software components to a hardware architecture goes beyond the simple task of mere allocation, and onto the generation of a deployment and corresponding schedule, which satisfies different constraints, be it related to safety, resource or cost.

The work presented in this paper is a part of ongoing research efforts into design space exploration (DSE), which aims at facilitating the seamless development of safety-critical applications in the embedded domain. DSE is defined as a process of systematically finding a solution from a set of possible designs, w.r.t. a set of given constraints [1]. Building on our pre-existing approach for the efficient generation of embedded system architectures with multi-criteria optimization (e.g., memory usage, power consumption, bus loads), we expanded the approach to include safety attributes, such as hardware failure metrics, but also practical aspects of real-time development, such as HW costs. The presented approach computes task and message schedules that are optimized with respect to a global discrete time base. As a part of the solution, the approach generates an optimized assignment of tasks to computation resources (cores). The approach is integrated into the AutoFOCUS3 (AF3) tool-chain, as presented in [2].

This paper presents the approach, and demonstrates its usefulness using examples which illustrate the effect of different criteria on deployment synthesis and how the approach can be used to optimize the generated valid solution for one or more criteria. Prior to this work, there was no support for generating and optimizing deployments in AF3 while simultaneously satisfying safety metrics, resource constraints and hardware costs.

Furthermore, AF3 integrates multiple dedicated architectural views, covering not only logical (software) and technical (hardware) architectures, but also requirements and safety case expression. Therefore, the method presented here can be integrated into a seamless development approach that allows arguing evidence on a formal basis, as well as reusing argument structures and reducing the costs of component as well as design (re-)certification for safety-critical applications.

A. *Related Work*

Approaches where computer-aided methods are used for system analysis and optimization are not new, having been proposed as early as in the late fifties, such as the optimization approach for register transfer systems towards cost and performance presented in [3].

Recent works address higher levels of abstraction, such as the mapping of software to hardware and the corresponding scheduling [4]; memory optimization [5]; or finding suitable platform architectures [6][7][8]. In this paper we address the joint problem of finding an optimized platform, deployment and schedule w.r.t. to safety, cost, performance and resource consumption.

The applicability of solvers for finding deployments and schedules was shown in [9] and [2]. The possibility of using various solvers for DSE problems was further discussed in [10] and [11], which – similarly to our work – use a solver to find valid or optimized solution.

Furthermore there are approaches, which take safety into account. [12] – for instance – combines FTA analysis and optimization techniques to find optimal system configurations; [13] optimizes automotive architectures towards cost and safety; and [14] uses a brute-force algorithm to generate safety-compliant deployments for avionic systems. Compared to those approaches, the optimization problem we are considering is more complicated (higher number of criteria and/or degrees of freedom), and furthermore we integrate our approach into a model-based framework, which makes it more usable for practitioners.

There are a number of frameworks with Design Space Exploration capabilities, which were proposed over the years. Some of them use their own DSLs to specify DSE Problems, such as FORMULA [15] and AAOL [16]. Others use well-established formalisms, such as EAST-ADL[17], UML [18] and AADL[19]. We integrated our approach in the AutoFOCUS3 Framework [1], which is based on the FOCUS methodology [20] and was shown to be compliant to AUTOSAR [2] in [21], leveraging the framework's seamless development capabilities.

The safety metrics used in the multi criteria optimization have their foundations in the ISO26262 automotive functional safety standard, and investigations into architecture benchmarking in the ITEA2 SAFE research project[3] published in [22].

Due to the seamless nature of AF3, our approach demonstrates how an integrated model-based framework can pragmatically provide formal support for practical problems in the development and certification of safety-critical systems in compliance with the relevant standards – in our case ISO26262.

B. *Structure of Paper*

Chapter II gives an overview of background information building the fundamentals of this paper. In III the model-based CASE tool AutoFOCUS is presented. In this tool, we integrate our proposed approach from chapter IV. We evaluate our work using a non-trivial case-study in chapter V and finally we conclude in VI.

II. Background

This section introduces briefly some concepts necessary for understanding the rest of the paper.

A. *SMT*

A satisfiability (SAT) problem consists of variables represented in a formula, with the goal being to determine whether there exists an assignment of variables that makes the whole Boolean formula satisfiable [23]. Satisfiability modulo theories (SMT) generalize boolean satisfiability by adding equality reasoning, arithmetic, arrays, quantifiers and other first order theories. SMT solvers are tools for deciding the satisfiability of formulas in those theories [24], which can be used to solve various classes of problems, such as software and hardware verification; test case generation; planning; or scheduling and deployment [25, Ch. 2, p. 89ff]. Solvers may be designed to not only be capable of determining whether a formula is unsatisfiable or not but also point to the set of clauses which are not satisfiable, called unsatisfiable cores or UnSAT Cores.

B. *Safety*

The work presented here was carried out within the SAFE research project, which targeted the model-based development of software architectures for safety-critical applications in the automotive domain. Hence, we focus on metrics identified in the ISO26262 automotive safety standard. These include ASIL (Automotive Safety Integrity Level), as well as hardware failure metrics identified in clauses of part 5 of the standard. The proposed metrics include PMHF (probabilistic measure of random hardware failures) the maximum probability of violating a top level safety goal, shown mapped to corresponding ASIL levels in Table 1 - and FRC (failure rate classes) an evaluation of each cause of safety goal violation. These metrics are explained in more detail in [22].

[1]http://af3.fortiss.org/

[2]http://www.autosar.org/

[3]http://www.safe-project.eu/

ASIL Level	Probability of random hardware failures (PMHF)
D	$< 10^{-8}h^{-1}$
C	$< 10^{-7}h^{-1}$
B	$< 10^{-7}h^{-1}$
A	$< 10^{-6}h^{-1}$

TABLE I: Target values for hardware architectural metrics

C. *Multi-objective Optimization*

As an optimization problem, we consider the search of a good (optimized) or best (optimal) solution of a decision problem among a set of alternatives, assuming the existence of a set of objectives, according to which the quality of the alternatives can be measured [26]. Cases in which an optimization problem does not only have one objective, but several ones are called: multi-objective optimization, and these objectives are often contradictory. Therefore, in most cases of multi-criteria optimization problems, there is no single optimal solution for the problem under consideration, but a set of solutions, where each is optimal to a subset of objectives. This set of solutions is called Pareto-efficient or the *Pareto front*.

III. AutoFOCUS3

AutoFOCUS3 (`http://af3.fortiss.org`) is a research CASE tool that allows modeling and validating concurrent, reactive, distributed, timed systems on the basis of formal semantics [27]. Furthermore – in [21] – it was illustrated how AF3 modeling concepts fit into concepts provided by AUTOSAR.

A. *Levels of Abstraction*

An AF3 system model is divided into several models that provide different levels of abstraction, while supporting different views on the system model, e.g., from the model-based requirements view down to the hardware-related platform view.

The *Requirements Specification and Analysis View* provides for requirements specification, documentation, and analysis of the requirements of a system.

The *Logical Architecture View* of a system is defined by means of components communicating via message passing through typed channels, using a clearly defined model of computation. Messages exchange is synchronized with respect to a global, discrete time base. Components can directly implement behavior or consist of other components that do so.

The *Technical Architecture View* describes a hardware topology that is composed of hardware units, e.g. CPUs of a Multi-Core Board, hardware ports (sensors or actuators), buses and a shared-memory component.

Furthermore, AF3 supports so-called integrated or weaving models, which hold information from different levels of abstraction and thereby connect them. The deployment view is one example of such an integrated model, which maps elements from the *component* to elements of the *technical architecture*. This provides traceability between modeling artifacts [1].

Another example is the AF3 support for a safety-case view, which is supported by integrated models, linking the safety case elements to all the other AF3 artifacts in support of the argument structure.

B. *Design Space Exploration in AF3*

A valid deployment and schedule pair for any system under consideration has to fulfill certain constraints, for instance, precedence constraints or resource utilization (two tasks can't run in parallel on the same node). These constraints can be formalized in such a way that they can be used as input for a state-of-the-art SMT solver. In this sub-section we provide a small overview of work done in [2], which we extend in the following sections.

The approach uses the integrated deployment model, which combines information from the logical and technical architecture (cf. III-A). From the logical architecture this model contains a set of logical components (which, in this context, we call tasks) $T = t_0, t_1, ..., t_n$ passing messages $M = m_0, m_1, ..., m_k$ via channels. This communication structure results in dependencies between the tasks. Those dependencies can be used to derive constraints, such as the execution order of tasks, for the schedule generation.

Additionally, this model contains information from the technical architecture, such as a set of computational resources (nodes) N, a set of buses B, and in some cases memories MEM. Nodes are used to store and execute the tasks from the logical architecture, whereas buses and memories are used to exchange messages between the tasks.

By encoding this information in a SMT solver, it is possible to synthesize a valid deployment and schedule pair, if they exist. Furthermore, it is possible to produce not only valid but also optimized solutions (cf. [2] or [28]), using a meta-search on top of the SMT solver, which manipulates constraints – max. execution time, for instance–after every SMT request.

IV. Safety-oriented Deployment and Scheduling in AF3

In this section we present how using a SMT solver makes it possible to synthesize safety oriented deployments, which are either valid or optimized w.r.t. to certain criteria. Furthermore we discuss how we integrated this approach in AutoFOCUS3.

A. *Artefact properties in AF3*

As already explained in III, AutoFOCUS3 view on the system is divided in different levels of abstraction. Each of these levels contains a set of artifacts (i.e. ECU in Technical Architecture). Using the "annotation" concept of AF3 it is possible to assign properties to each artifact (i.e. Failure Rate to an ECU). For our approach a set of safety and resource related properties were needed, which partly already existed and partly were added by us.

In context of safety we added the possibility of annotating PMHF values for ECUs and busses. This value is then

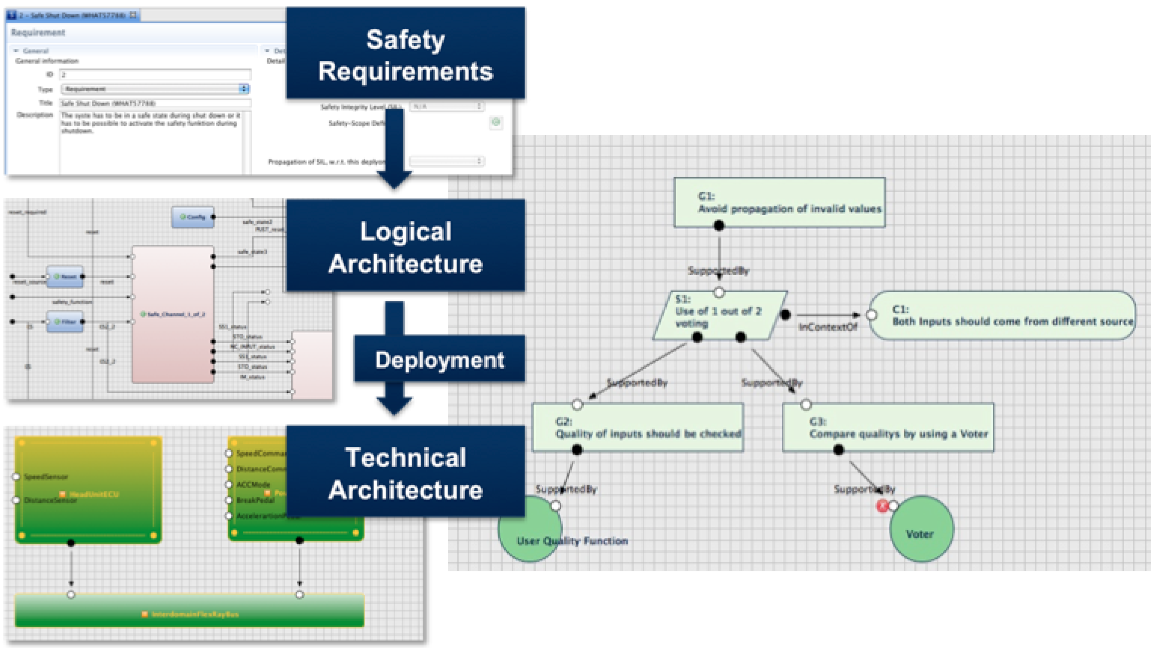

Safety Requirements
Logical Architecture
Deployment
Technical Architecture
G1: Avoid propagation of invalid values
SupportedBy
S1: Use of 1 out of 2 voting
InContextOf
C1: Both inputs should come from different source
SupportedBy
SupportedBy
G2: Quality of inputs should be checked
G3: Compare qualitys by using a Voter
SupportedBy
SupportedBy
User Quality Function
Voter

Fig. 1: Seamless model-based development in AF3

translated to ASIL according to table I. This way it is more convenient to compare with the safety integrity level values of the logical components. The possibility of assigning (A)SIL values to logical components was already given.

Furthermore we added the possibility of assigning energy consumption, memory and cost to the hardware elements as well as memory consumption and taswk duration to the logical components.

B. Scheduling and Deployment Constraints

The constraints – introduced in this sub-section – are already formalized. Together with instructions, such as in [29] or examples such as in [2], these formalized constraints provide enough information to be encoded as a SMT problem.

1) PMHF Constraint for ECU: This constraint states that a task $t \in T$ can only be mapped on a node $n \in N$, if the PMHF value of the node n is compliant to the ASIL value of the task t according to table I. Let $\Phi : T \mapsto N$ be a function that allocates tasks to nodes, then the constraint can be formalized as follows:

$$\forall t \in T', \text{where } T' = \{t' | \Phi(t') = n\} \rightarrow t.asil \leq n.asil \quad (1)$$

2) PMHF Constraint for BUS: As defined in [2], if two tasks $t_i t_k \in T$ communicate with each other and are deployed on different nodes, such that $\Phi(t_i) \neq \Phi(t_j)$, they have to communicate over a bus $b \in B$, which also has a PMHF value. Furthermore, we respect ASIL propagation, which means that tasks can only exchange messages via bus, if the bus is sufficiently reliable, as indicated by its PMHF value. This constraint affects the allocation of communciating tasks. If t_i sends a message to t_k and those tasks are mapped on different nodes, the constraint can be formalized as follows:

$$b.asil \geq t_k.asil \quad (2)$$

3) Memory Constraint: In real-life embedded systems resources, such as memory, are limited. We assume that each hardware node n offers a certain amount of memory that can be used by the tasks t allocated to it, and must not be exceeded. To define this constraint, we first have to sum up the memory which is used by each node:

$$\forall t \in T', \text{where } T' = \{t' | \Phi(t') = n\} \rightarrow$$
$$n.used\_memory = \sum t.mem \quad (3)$$

Having determined the memory used by each node, we can now ascertain that the used memory does not exceed the available memory ($n.ram$):

$$n.used\_memory \leq n.ram \quad (4)$$

4) Maximum Number of Nodes: In some cases it is interesting to know whether the deployment of tasks is possible on less nodes than currently used or available. To define a constraint limiting the number of used nodes, we first have to find out which nodes are currently *used* (has tasks deployed on it). We can formalize this as follows:

$$n.used = \begin{cases} 1, & \exists t \in T : \Phi(t) = n \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

Next, we identify how many nodes are currently used:

$$total\_used\_nodes = \sum_{n}^{N} n.used \quad (6)$$

Finally, we can formalize the constraint as follows:

$$total\_used\_nodes \leq max\_nodes \quad (7)$$

5) Cost: Cost is a very important factor in industrial projects. This is especially true in context of automobile industry, since even a small saving in component costs can impact a lot due to the large numbers of produced cars. As already discussed, PMHF values of hardware components can be mapped to ASIL values, but also to component costs, under the assumption that hardware with more stringent failure rates is also more expensive to build. It is thus relevant to define a separate constraint to limit component costs, without violating any other constraints. We first need to know which nodes are used in the current deployment, the formalization of which has already been shown in (eq. 5). Based on this knowledge, it is possible to calculate the total cost of the system:

$$total\_cost = \sum_{n}^{N} n.cost \times n.used \tag{8}$$

Subsequently we restrict the maximum cost of the system as follows:

$$total\_cost \leq max\_cost \tag{9}$$

6) Power Constraint (Energy Constraint): Energy consumption is another important factor in embedded systems and hence we generate deployments that take energy efficiency into account. In our simplified energy model we assume that each node consumes energy only as long it is active (non-idling). Therefore the non-idling time can be calculated as follows:

$$n_{non\_idle\_time} = \sum\nolimits_{t \in T'} t_{duration} \quad \text{where } T' = \{t' | \Phi(t) = n\} \tag{10}$$

Since power is defined as energy over time the energy consumption of a node n can be formalized with the following equation:

$$n.energy = n.power \times n.non\_idle\_time \tag{11}$$

Using (eq. 12) it is now is possible to calculate the total energy consumption of the system. This value again can be constrained by the maximum allowed energy consumption (eq. 13).

$$total\_energy\_consumed = \sum_{n}^{N} n.energy \tag{12}$$

$$total\_energy\_consumed \leq max\_energy \tag{13}$$

C. Pareto-efficient Deployments

In the previous sub-section we presented the constraints, which – in scope of this work – are used to synthesize valid deployments with certain characteristics. In the following sub-section we discuss how to find optimal (or at least optimized) deployments w.r.t. certain criteria.

The problem of generating the pareto-optimal deployments can be formulated in terms of objectives to be optimized and constraints to be satisfied (as shown below). The formulation is followed by a brief explanation of the objectives and an introduction of the proposed optimization algorithm.

Minimize

$$number\_of\_nodes$$
$$memory\_per\_node$$
$$total\_sil$$
$$e2e\_latency$$

such that

$$number\_of\_nodes \leq max\_nodes$$
$$memory\_per\_node \leq max\_memory$$
$$total\_sil \leq upper\_bound\_sil$$

PMHF constraint holds

The values of variables max_nodes, max_memory and $upper\_bound\_sil$ are entered by the user, if they choose to generate pareto-optimal deployments. The PMHF constraint are also needed to be satisfied while generating the pareto-optimal deployments. Since we are aiming at minimizing the total SIL of the hardware architecture, we do not use *cost constraint* and *energy consumption constraint* explicitly.

1) Optimization Criteria: In [30], it was illustrated that participants of the survey – if it comes to system design optimization – are interested in such criteria as safety, timing, resource usage, energy consumption and cost.

Following this line of argumentation we implemented a multi-objective optimization algorithm which takes number of used nodes, used memory per node, as well as hardware costs and energy consumption as optimization criteria. For simplicity, we assumed hardware costs and power consumption to be directly correalted to the ASIL value, and thus used total ASIL as a proxy optimization criterion.

Furthermore, we took timing (end-to-end latency) into account using a solution, which, while not being Pareto-optimal according to the definition in II-C, was still optimized as explained in IV-C2.

2) Optimization Algorithm: Our algorithm works in three steps: reduction of the search-space, generation of all valid solutions for the reduced search-space, and elimination of dominated solutions, i.e., those solutions which are superseded by the pareto front.

In the first step, our approach reduces the search-space by calculating lower bounds for the criteria *memory per node* and *number of nodes*. Consider a scenario wherein the logical architecture contains 5 components and each component requires 10 units of memory. Let the values of variables *max_nodes* and *max_memory* (as entered by the user) be 4 and 20 respectively. It is clear that there will be no possible solution if the number of nodes in the technical architecture is less than 3, since the total memory needed to accommodate the 5 logical components is 50 and maximum memory allowed per node is 20. Similarly, the memory per node can be no less than 20 (assuming memory per node is a multiple of 10). As the result of this step we get a set of possible configurations in terms of a range for *number of nodes* i.e. $[min\_nodes, max\_nodes]$ and for *memory per node* i.e $[min\_memory, max\_memory]$.

In the second step, we generate valid solutions for all possible combinations of nodes and memory found in the previous step. This is illustrated in the algorithm 1 below.

Lines 2-4 shows that for each pair of nodes and memory in decision variable space, a constraint satisfaction problem

Algorithm 1 Valid Solutions

```
1: procedure ValidSolutions
2:     for $nodes$ in $min\_nodes$ .. $max\_nodes$ do
3:         for $memory$ in $min\_memory$ .. $max\_memory$ do
4:             $result \leftarrow CheckSAT(nodes, memory, upper\_bound\_sil)$
5:             if $result$ is $SAT$ then
6:                 $solution \leftarrow parse(result)$
7:                 store $solution$ to $UniqueSolutions$
8:                 $total\_sil \leftarrow extractSIL(solution)$
9:                 while $result$ is SAT do
10:                    $total\_sil \leftarrow total\_sil - 1$
11:                    $result \leftarrow CheckSAT(nodes, memory, total\_sil)$
12:                    if $result$ is $SAT$ then
13:                        $solution \leftarrow parse(result)$
14:                        store $solution$ to $UniqueSolutions$
```

is created which is solved by the Z3 SMT solver. The $checkSAT()$ function formulates the problem in Z3 syntax, where the parameters $nodes$, $memory$ and $upper\_bound\_sil$ defines the upper limits for assertions used in node usage, memory per node and total sil constraints, respectively . Line 5 checks if the constraint satisfaction problem is satisfiable. In case it is satisfiable, the output of the Z3 solver is parsed to fetch *number_of_nodes*, *memory_per_node*, *total_sil* and *e2e_latency*. These variables are combined into a $solution$ (line 6) and stored into a set of unique solutions (line 7), as we do not want to duplicate the set of valid deployments. We then try to optimize the *total_sil* by reducing it in steps of 1 (line 10) and check the satisfiability of new constraint satisfaction problem (line 11). The reduction of *total_sil* continues until Z3 cannot find a solution to the constraint satisfaction problem (line 9).

In the last step, the algorithm eliminates the solutions which are dominated by other solutions w.r.t. number of nodes, memory per node, total ASIL and thereby builds a Pareto front. As mentioned earlier our approach chooses a solution with the lowest end-to-end latency from the set of valid solutions, which were produced in step 2. For that reason we call this solution not optimal but optimized.

D. Integration in AutoFOCUS3

In scope of this work we integrated our synthesis approach together with the constraints and the optimization algorithm (introduced in the previous sub-sections) in the AutoFOCUS3 development methodology.

A typical development process in AF3 could look as follows. An engineer, designing a safety-critical system in AutoFOCUS3, ideally starts with the definition of requirements. Among other things in this phase, he will identify a set of safety goals which the system under development will have to fulfill. One of the safety goals could be that safety-critical software should not be compromised due unsafe hardware it is running on, which corresponds to the constraints 1) and 2) from IV-B.

In the next step, the engineer starts to design the structure and the behavior of the system in the logical architecture (cf. III-A). After that he can assign certain properties (cf. IV-A), such as Safety Integrity Level, to the elements of this level of abstraction. At this step it possible either to continue with the design of the technical architecture (i.e. the technical platform of the system is already fixed) or use the optimization approach from IV-C to find an optimized technical architecture together with the corresponding deployment and schedule. Regardless which option the engineer takes at some point he will end up with a logical and a technical architecture. Now he can use either the constraints from IV-B or the approach from IV-C to find either valid or optimized deployments and schedules w.r.t. the constraints he is interested in (cf. fig. 2).

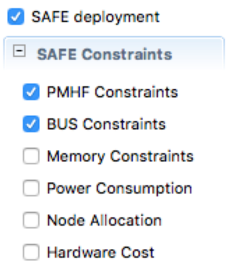
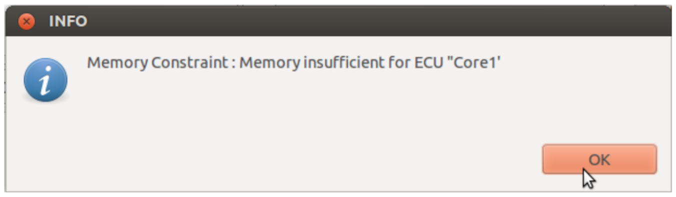

☑ SAFE deployment
⊟ SAFE Constraints
☑ PMHF Constraints
☑ BUS Constraints
☐ Memory Constraints
☐ Power Consumption
☐ Node Allocation
☐ Hardware Cost

Fig. 2: Choosing constraints in AF3

The deployments and schedules which are found during this Design Space Exploration activity are valid towards certain constraints (such as 1) and 2) from IV-B). Therefore, the results of this DSE run can now be used as an evidence in a safety case to prove that the safety goal is reached by design. The integration of safety cases into AF3 development process was demonstrated in [31].


INFO

Memory Constraint : Memory insufficient for ECU "Core1"

Fig. 3: Finding the unsatisfiable constraints

Given the potential complexity of the optimization problems addressed by our approach, it is possible to define the constraints so tightly that no solution can satisfy them all. For instance, the provided pool of ECUs – even if all of them were used in combination – does not provide enough memory to run all the software components. It is thus just as useful to support the user by providing guidance as to which constraints could not be satisfied. For this case we integrated the Z3s UnSAT Core functionality (cf. sec. II-A), which identifies unsatisfiable constraints for the current DSE problem. As seen in figure 3, this feature points the user exactly to the constraint which is not satisfiable and, hence, needs to be adjusted.

V. Evaluation

A. Description of the Case Study

We evaluated our approach using an industrial-like case study, which was created in the scope of the SAFE Project. This case study was modeled in AutoFOCUS3 and consists

ASIL Level	Components
ASIL D	c1, c2, c3, c4, c5
ASIL C	c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17
ASIL B	c18, c19, c20, c21, c22, c23, c24, c25, c26, c27
ASIL A	c28, c29, c30, c31, c32, c33, c34, c35, c36, c37, c38, c39
QM	c40, c41, c42, c44, c45, c46, c47, c48

TABLE II: Logical Architecture of the Case-Study

of both a logical and technical architecture. The logical architecture consists of 48 components, connected by 65 channels. The components have different ASIL levels, as described in table II.

The technical architecture consists of 9 nodes connected by a bus. Each hardware component was assigned a predetermined PMHF value (and ergo ASIL compliance) and cost (cf. table III). These values can, of course, be modified by the user to represent different hardware characteristics.

ECUt	1	2	3	4	5	6	7	8	9
PMHF	0.1	0.1	0.3	0.4	0.6	0.6	0.8	0.9	1.0
ASIL	D	D	C	C	B	B	A	A	QM
Cost	10	10	8	8	5	5	3	3	1

TABLE III: Technical Architecture of the Case-Study

B. *Cost versus Safety*

In our investigations, we analyzed how manipulating the criteria - both individually and collectively - affects the deployment and schedule generation process. The investigation is too large to list in detail here; more results are provided in [22]. To demonstrate the approach, we will focus on one example: how cost reduction affects safety-oriented deployments; an optimization problem reflecting design trade-offs, under the assumption that hardware with more stringent failure rates is more expensive to build. The results are shown in chart form in the next figure. The chart (cf. fig. 4) shows possible

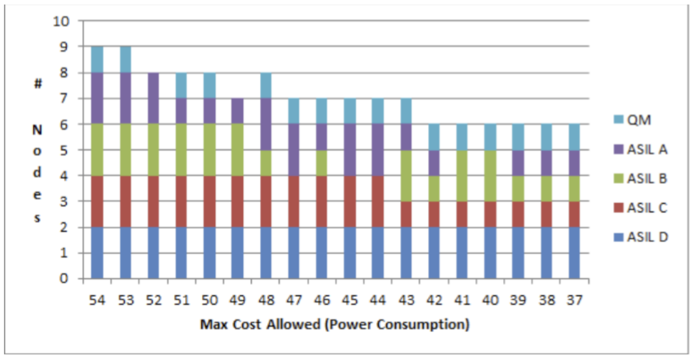


Fig. 4: Effect of Cost Constraint (non-uniform cost) on deployment (ASIL-D BUS)

deployments to satisfy the multi-criteria problem, while optimizing for power consumption, total cost of hardware, and total number of nodes. The results show that it is possible to satisfy the problem with various configurations and allows the user to choose which criteria are more relevant, and hence which deployment – among the multiple *valid* ones – is *most suitable* to their needs.

C. *Evaluation of the Optimization Approach*

In our second experiment, we investigated how long our approach would take to find a set of non-dominated, i.e., optimized, solutions. As input we used the case-study from section V-A. The search space was restricted using the following intervals: 1 to 9 nodes; 1 to 36 as total ASIL; and 10 to 100 MB memory per node (using a 10 MB iteration step). As discussed in section IV-C, the end-to-end latency does not have to be restricted because our approach picks the solution with the best end-to-end latency from all the solutions it finds. The total ASIL constraint represents the hardware cost factor, by assigning a high number to more capable hardware nodes.

Number of nodes	Memory per node	E2E latency	TotalSIL
5	100	263	10
7	70	292	14
6	80	248	16
6	80	257	14
6	90	210	11
8	60	269	17
6	80	281	13
8	90	192	13

TABLE IV: Pareto deployments for industry-like use case

In our experiment we used the Z3 solver with a 6 hours time limit for each evaluation. This means, that after this time period the Z3 solver times out with the best solution so far, which might be not the overall optimal solution. The result of this experiment, presented in table IV, was calculated in 126 hours. Given the size of the case study and the degrees of freedom under consideration (deployment, schedule, memory, SIL distribution, timing) we think that this time period is reasonable and feasible in a practical setting.

Most importantly, the approach allows for the explicit optimization for selected criteria, the impact of which can be seen in the last tables. One can have different deployments on the same as well as different numbers of nodes with different effects on resource usage, system attributes, and hardware costs. For instance, if the number of nodes is the absolute limiting factor, then one can choose the first deployment with 5 nodes. If however, one can accommodate a design with 6 hardware nodes, then it is possible to choose a solution optimized for Memory consumption and hardware costs, which also provides a comparatively low latency value. More details on the different effects of optimizing for different criteria are given in [22].

Above all, these outcomes are based on formal methods, they are deterministically reproducible, and form a solid foundation for safety assurance and correct/safe-by-construction solutions.

VI. Conclusion and Future Work

In the engineering of reliable and safe automotive systems, the process of mapping software to hardware is an essential design step. In this paper we presented a design space exploration approach for multi-criteria optimization of the deployment problem, formalizing constraints relevant for system development according to ISO26262.

Not limiting our interest to generating merely valid solutions, but in good ones, we developed a multi-objective optimization algorithm, which synthesizes solutions pareto-optimized for safety, resource usage, timing and any other constraints the user defines. A state-of-the-art SMT solver is used in conjunction with the formalized constraints to find valid solutions, which satisfy the constraints, while a meta-search on top of the SMT solver provides the optimized solutions. In our example, we derived 6 constraints and implemented an optimization algorithm for 4 criteria, with ongoing work to give users more freedom to define their own constraints and criteria. Above all, the generated results are based on formal methods, they are deterministically reproducible, and form a solid foundation for safety assurance and correct/safe-by-construction solutions.

Our approach demonstrates the feasibility and effectiveness of using formal methods to generate correct solutions for safety-critical applications under real-world scenarios, increasing the confidence and validity of safety evidence for certification.

Solutions to software engineering problems are most useful when bolstered by tool-support; we integrated our work in a model-based framework, called AutoFOCUS3. Due to the seamless nature of AF3, our approach demonstrates how an integrated model-based framework can pragmatically provide support for practical problems in the development and certification of safety-critical systems in compliance with the relevant standards, and just as importantly, how formal methods can easily be made more accessible to practitioners without a formal background.

References

[1] S. Voss, S. Zverlov *et al.*, "Design space exploration in autofocus3 - an overview," in *IFIP First International Workshop on Design Space Exploration of Cyber-Physical Systems*. Springer, 2014.

[2] S. Voss and B. Schaetz, "Deployment and scheduling synthesis for mixed-critical shared-memory applications," in *Engineering of Computer Based Systems, 2013 20th IEEE International Conference and Workshops on the*, 2013, pp. 100–109.

[3] M. Breuer, "Recent developments in the automated design and analysis of digital systems," *Proceedings of the IEEE*, vol. 60, no. 1, pp. 12–27, Jan 1972.

[4] E. Kang, E. Jackson *et al.*, "An approach for effective design space exploration," in *FOCS'10*. Springer-Verlag, 2011, pp. 33–54.

[5] S. Kuenzli, "Efficient design space exploration for embedded systems," Ph.D. dissertation, ETH Zurich, April 2006.

[6] E. B. Fernandez and B. Bussell, "Bounds on the number of processors and time for multiprocessor optimal schedules," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 745–751, 1973.

[7] S. Prakash and A. C. Parker, "Synthesis of application-specific heterogeneous multiprocessor systems," in *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2. ACM, 1992, p. 434.

[8] M. Thompson, H. Nikolov *et al.*, "A framework for rapid system-level exploration, synthesis, and programming of multimedia mp-socs," in *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, New York, NY, USA, 2007, pp. 9–14.

[9] A. Metzner, Franzle *et al.*, "Scheduling distributed real-time systems by satisfiability checking," in *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*. IEEE, 2005.

[10] S. Kugele, L. Dieudonné, R. Popa, and H. Eckardt, "On the Deployment Problem of Embedded Systems." *In 13th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE'15)*, no. i, 2015.

[11] T. Saxena, "A generic framework for design space exploration," Ph.D. dissertation, Vanderbilt University, 2012.

[12] F. Ortmeier and W. Reif, "Safety optimization: a combination of fault tree analysis and optimization techniques," *International Conference on Dependable Systems and Networks, 2004*, pp. 1–8, 2004.

[13] M. S. Dhouibi, L. Saintis, M. Barreau, and J.-M. Perquis, "Safety driven optimization approach for automotive systems," in *Reliability and Maintainability Symposium (RAMS), 2015 Annual*. IEEE, 2015, pp. 1–7.

[14] R. Hilbrich and L. Dieudonné, "Deploying Safety-Critical Applications on Complex Avionics Hardware Architectures," *Journal of Software Engineering and Applications*, vol. 06, no. 05, pp. 229–235, 2013.

[15] E. Kang, E. Jackson, and W. Schulte, "An approach for effective design space exploration," *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pp. 33–54, 2011.

[16] S. Kugele and G. Pucea, "Model-based optimization of automotive E/E-architectures," *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis - CSTVA 2014*, no. MAY 2014, pp. 18–29, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2593735.2593739

[17] M. Walker, M. O. Reiser, S. Tucci-Piergiovanni, Y. Papadopoulos, H. Lönn, C. Mraidha, D. Parker, D. Chen, and D. Servat, "Automatic optimisation of system architectures using EAST-ADL," *Journal of Systems and Software*, vol. 86, no. 10, pp. 2467–2487, 2013.

[18] B. Selic and S. Gérard, *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Elsevier, 2013.

[19] a. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, "ArcheOpterix: An extendable tool for architecture optimization of AADL models," *Model-Based Methodologies for Pervasive and Embedded Software, 2009. MOMPES '09. ICSE Workshop on*, pp. 61–71, 2009.

[20] M. Broy and K. Stølen, *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Springer Science & Business Media, 2012.

[21] S. Z. B. Schaetz, S. Voss, "Automating design-space exploration: Optimal deployment of automotive sw-components in an iso26262 context," in *Design Automation Conference (DAC), 2015 52st ACM/EDAC/IEEE*, 2015.

[22] SAFE-E, "Deliverable d4.4b: Final version of plug-in for safety and multi criteria architecture modeling and benchmarking," The SAFE-E Consortium, Tech. Rep., 2014.

[23] J. Gu, "Local search for satisfiability (sat) problem," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 4, pp. 1108–1129, 1993.

[24] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[25] F. Van Harmelen *et al.*, *Handbook of knowledge representation*. Elsevier, 2008.

[26] M. Ehrgott, *Multicriteria optimization*. Springer, 2005, vol. 2.

[27] A. Bauer, M. Broy *et al.*, "Automode - notations, methods, and tools for model-based development of automotive software," in *SAE International*, 2005.

[28] S. Zverlov and S. Voss, "Synthesis of pareto efficient technical architectures for multi-core systems," in *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*. IEEE, 2014, pp. 366–371.

[29] N. Eén and N. Sorensson, "Translating pseudo-boolean constraints into sat," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.

[30] P. Diebold, C. Lampanosa *et al.*, "Practitioners' and researchers' expectations on design space exploration for multicore systems in the automotive and avionics domains – a survey," in *EASE Proceedings*. ACM Digital Library, 2014.

[31] S. Voss, B. Schätz, M. Khalil, and C. Carlan, "Towards modular certification using integrated model-based safety cases," in *proc. VeriSure: Verification and Assurance Workshop*. Citeseer, 2013.

Comparing several candidate architectures variants : An Industrial Case Study

Sébastien Madelénat,
Christophe Labreuche
and Jérome Le Noir
Thales Research & Technology, France
firstname.name@thalesgroup.com

Grégory Gailliard
Thales Communications & Security, France
firstname.name@thalesgroup.com

***Abstract*—In system and software engineering, the analysis of architectural variants is most of time irrational and manual. The most common approach for comparing variants is comparing results for each variant evaluation. Most advanced approaches available in architecture evaluation are suffering from three principal weaknesses: the absence of criteria elicitation method, no representation of real-life strategies and no explaination of the outcomes. This paper relates experiments of a MCDA tooled method addressing these weaknesses. The experimentation is supported by an industrial use case consisting in selecting the best platform for an handheld Software-defined Radio. Its architecture description is formalised with model-based design tools. As a result we conclude the experimented approach provides sharper results than classic approach on the class of decision problem exposed by avoiding false positives. This approach seems to be promising to improve the confidence in our Decision Analysis Report and their quality in terms of argumenting the reasons of a decision.**

Keywords: System engineering, Decision model, Tradeoff, Multi Criteria Decision Analysis

I. Introduction

In system and software engineering, the analysis of architectural variants is most of the time subjective and manual. The justification of a variant is seldom based on the assets, the flaws and strengths of the different options. Ideally, assessing or comparing several candidate architectures (variants) should be based on some decision criteria – corresponding to a Multi- Criteria Decision Aiding (MCDA) problem. Among the classic hand-made "Decision Analysis Report", widely used in the industry, the industrial methods and tools state-of-the-art presents several weaknesses.
This paper presents an industrial practical experience based on a MCDA tooled method for evaluating and comparing several candidate deployments of a Software-defined Radio application on hardware platforms[1] in a Model-based system engineering (MBSE) context.

A. Needs for decision aid to select the best candidate alternative

In [11] the empirical demonstration has been made that variability in the description model of the architecture allows to easily construct a set of candidate architecture descriptions. Then one aims at finding the options that best fits with the needs of the various stakeholders. This choice problem can be formulated as the maximization of a set of decision criteria. The difficulty is that the decision criteria are usually numerous and conflicting. One may indeed have performance criteria versus cost criteria which cannot be met both at the same time. The difficulty is manifold. First of all, the decision criteria are described by metrics and one first needs to identify a set of relevant metrics. Then, the selection of one alternative among several on the basis of a set of metrics is complex since there are commensurateness issues (combine "apples with oranges" as the metrics are given in different units), and one aims at making arbitrage between the metrics.
Nowadays, in system and software engineering, practices make this analysis most of the time empiric and irrational ([9], [8], [1], [5] and [7]) as described in subsection II-A. One may say deciding is an essentially irrational activity [12]. Therefore it is not very surprising to find irrational analysis practices. Rational analysis requires providing arguments about choices. MCDA approaches allow this. Moreover the justification of the choice of an alternative is very seldom based on the assets and flaws of the different options. Assessing or comparing several candidate alternatives on the basis of some decision criteria corresponds to a Multi-Criteria Decision Aiding (MCDA) problem. The approach consists in constructing an explicit multi-criteria decision model. The main benefits of explicitly constructing such a model are to reach objectivity in the analysis, come up with a recommendation from a well-established methodology, and the possibility to justify the results. The model is elicited from interviews with the stakeholders and expresses the preference of these actors.

The Section 2 presents an analysis of existing approaches in engineering and the flaws induced, the Section 3 presents our approach, the Section 4 describes the experimental use case and the evaluation preparation, finally the Section 5 concludes on the results and discusses about the proposed approach applicability.

[1]The research activities were conducted in the context of ITEA2-MERgE (Multi-Concerns Interactions System Engineering, ITEA2 11011), a European collaborative project with a focus on safety and security

II. Analysis of the existing Multi-criteria Approaches in Engineering

The problem of identifying a set of relevant metrics has been performed in different domains such as performance ([16]), availability ([14]), modifiability ([13]). Moreover, the Software Engineering Institute (SEI) has developed the Architectural Trade-off Analysis Method SM (ATAM SM) and validated its usefulness in practice ([15]). ATAM aims at making rational choices among competing architectures. It bases its criteria elicitation on stakeholders scenario over the system in consideration. The ATAM uses stakeholders' perspectives to produce a collection of scenarios that define the qualities of interest for the particular system under consideration. Scenarios give specific instances of usage, performance and growth requirements, types of failures, and possible threats and modifications. Once the important quality attributes are identified, the architectural decisions relevant to each one can be analyzed with respect to their appropriateness. The ATAM was designed to make rational choices among competing architectures, based upon well-documented analyses of system attributes, concentrating on the identification of trade-off points. The ATAM also serves as a vehicle for the early clarification of requirements. As a result of performing an architecture trade-off analysis, enacting ATAM allows an enhanced understanding of, and confidence in, a system's ability to meet its requirements. It helps also eliciting a documented rationale for the architectural choices made, consisting of both the scenarios used to motivate the attribute-specific analyses and the results of those analyses. Compared to traditional MCDA methods, ATAM does not perform multicriteria analysis. It qualifies the link between the description of the architectures (e.g. through a feature model) and the criteria, thanks to concepts such as sensitivity points (a feature that has a large influence on at least one criterion) or trade-offs (a feature which has a positive impact on a criterion but at the same time a negative impact on another criterion). ATAM and MCDA approaches are complementary: ATAM can be used as a complement to a MCDA approach as well as ATAM can be completed by a MCDA approach.

A. Flaws of the existing MCDA methods used in system or software engineering

If one wishes to interpret the metrics and be able to assess the architectures or compare architectures, one needs to add further information. We have identified four levels of information that can be added:

Level 0. This is the case when one has no further information than the list of metrics, except the sense of preference on each metric. When one has only the list of metrics and the sense of variation on each metric, e.g. for a cost metric, the smaller the better. This ordering is not discriminating and does not allow selecting the best option. The Pareto ordering is useful when the number of criteria is small (usually 2 or 3), and when the number of options is large. These two assumptions are false in our context.

Level 1. One is assumed here to provide at least one reference value on each metric. The most usual reference value in engineering is the "target value" which is the budget value of the metric for which the associated requirement is met. Additionally, one may also add the "threshold value" which is the maximal or minimal value of the metric for which the utility of the system is seriously questioned. At this level, one may say whether a given criterion is either completely satisfied (comparison of the metric with the target value) or not satisfied at all (comparison of the metric with the threshold value) but no interpretation of the other values of the metric can be given. The methods on the engineering of requirements belong to this level but as the criteria are not combined, these approaches do not allow to perform a trade-off analysis between the attributes. This is however necessary when not all requirements can be met all together and some compromise shall be reached.

Level 2. On top of level 1, one assumes that a satisfaction function that provides the degree to which the criterion is satisfied for every value of the metric. This is usually described as a piece-wise affine function. This does not specify how to weigh up the values of the various criteria to provide an overall assessment of the alternatives.

Level 3. On top of level 2, one assumes that the way the different criteria shall be combined, is specified. This is classically done through weights assigned to the criteria. In order to make recommendations over the options that go beyond the Pareto ordering, one needs to go to Level 3. The most elaborate MCDA methods available in system engineering is the use of a weighted sum in a quantitative utility method. This allows computing an overall assessment of the candidate options. The existing approaches suffer from three main limitations.

No elicitation method. In the existing tools, the stakeholders enter directly the thresholds and the weights. The understanding of a stakeholder on the parameters of the model is always limited when they are out his expertise domain. The conclusions of the evaluation cannot be justified if the stakeholder provides directly the values of the parameters. Rather one shall ask the stakeholder to provide information that are confident with, e.g. example of decision or assessment, and deduce the values of the parameters. Hence some elicitation methods to construct the values of the parameters shall be used.

No representation of real-life decision strategies. Most of the aggregation functions used are the weighted sum. A weight is assigned to each criterion, quantifying the importance of criteria. The main asset of this model is to be easily understandable. However, this simple model suffers from several limitations. It assumes preferential independence between the criteria: the contribution of one criterion to the overall evaluation does not depend on the marks with respect to the other criteria. This independence is often not met due to the presence of interaction between criteria. A typical example is the presence of a veto criterion. In system analysis, one has often to consider high level consequences (such as operational, financial, human factor, and so on) at the highest level of analysis. The most important criterion concerns surely the operational aspects since the other ones are more or less non-functional. As a consequence, if the operational part is not well-satisfied there is no other well-satisfied criterion that can save the solution. There is no analyst that is happy of a solution that completely misses the mission but costs very little. This means that the operational aspects behave like a veto. This cannot be modeled by a weighted sum.

No explanation of the outcomes. The decision maker is usually the person who is responsible for the decision. He often has to explain his decision to other actors - for instance, peers, managers, executive board or shareholders. These actors have often no time to go into the technicality of the decision model. In order to convince them on the merits of the decision, a synthetic explanation needs to be given such as an argumentation about metrics interpretation based on criteria thresholds, weights and interactions.

B. Existing Metric Analysis Approaches in Engineering

In existing approaches, levels 2 and 3 are often put together. We will consider these levels together in the following. We present the review of the existing approaches used in architecture trades studies in model based system engineering among the levels of information described above.

Methods at Level 0: The construction of the Pareto frontier among the set of options is proposed in the ASDL (Aerospace Systems Design Laboratory) developed at Georgia Tech [9][8], and also in the ESTECO tool for multi-domain engineering[1].

Methods at Level 1: The methods on the engineering of requirements belong to this level but use only one level. The two objective and threshold levels are used in the capability-based approach developed by the US Department of Defense. In order to meet the future needs, the force transformation shall be analyzed according to seven concerns summarized under the acronym DOTMLPF (Doctrine, Organization, Training, Materiel, Leadership, Personnel and Facilities)[5].

Methods at Levels 2 and 3: Levels 2 and 3 are often put together. The most elaborate MCDA methods one can find in system and software engineering is the use of a weighted sum in a quantitative utility method. Some workflow simulation tools such as SIMUL-8 [2] are based on a MCDA weighted sum model. SIMUL-8 integrates a MCDA tool called VISA[3]. A simple MCDA approach based on a weighted sum has recently been integrated in the IBM Rhapsody tool to perform trade studies[4]. The Canadian Department of Defense has also developed its approach for Capability Based Planning. The evaluation for each capability domain is performed at the strategic, operational and tactical levels against six concerns summarized under the acronym PRICIE (Personnel, Research, Infrastructure, Concepts, Information et Equipment)[21]. An evaluation in the numerical scale [0,100] is performed on each concern and a relative importance between the concerns is expressed. This allows computing an overall assessment of the candidate options, using a simple weighted sum model.

III. Our approach

Myriad[17] is an experimental tool for MCDA developped at Thales Research and Technology France. It proposes advanced methods and tools for decision making. It has been successfully experimented in several operational cases in aiding in decision making.

Myriad proposes criteria elicitation tooled method for defining criteria relative weight and thresholds based on a technique among which an extension of MACBETH[10] to account for interacting criterion. Its aggregation function, the Choquet integral, is capable of taking into account important cases (veto, favor, complementarity among criteria) for simulating real decision strategies. Myriad proposes key features such as production of evaluation reports. These reports are about evaluation results explanation based on the evaluation model analysis, augmented with improvement recommendations sorted by potential score impact for each criteria. Moreover, when multiple evaluation are done using the same evaluation model, the report proposes evaluations comparison argumentation. Regarding ATAM, the presented method does not addresses the metrics identification or architecture description updates recommendations for improvement. For this latter part, in [20] Montmain et al. proposed an extension of the method presented in this paper. It consists in an optimization algorithm recommending feature variables updates that maximizes the overall criteria satisfaction based on an influence model relating the feature model to criteria. For these reasons we propose to experiment the capability of Myriad to go beyond limits of existing approaches.

A. Basic concepts

The three basic concepts in the evaluation tree are:

Metric "U". Usually, a metric is a numerical quantity to assess the level of one objective achievement. We will use the word "metric" in a broader sense (like attribute)[22]: instrument which synthesizes in qualitative or quantitative terms, certain information which should lay the foundation for a judgment of an alternative relative to certain of its characteristics, attributes or effects (consequence).

Criterion "C". A criterion is a specification of the preference that an individual has on the values of a metric relatively to a concern. This specification amounts to construct a function – called **utility function** – which returns for each value of the metric the relative performance level (goodness) which positions it on a preference scale. The underlying scale is often a numerical scale, such as the [0,1] interval in which the value 0 is judged unacceptable relatively to the concern of the criterion, and value 1 is judged perfectly satisfactory relatively to the concern of the criterion. One can give an absolute judgment on an alternative according to a criterion.

Aggregation[22] "A". This is a procedure that produces an evaluation of any alternative by taking into account, in a comprehensive way, the performance levels of the alternative according to the criteria corresponding to a set of concerns. There are often nested aggregations. The hierarchical organization of criteria in nested aggregations is due to done to group criteria according to similar concerns.

B. Concept related to utility functions

The construction of a utility function requires two elements. Firstly, an **interval scale** is needed. The evaluation

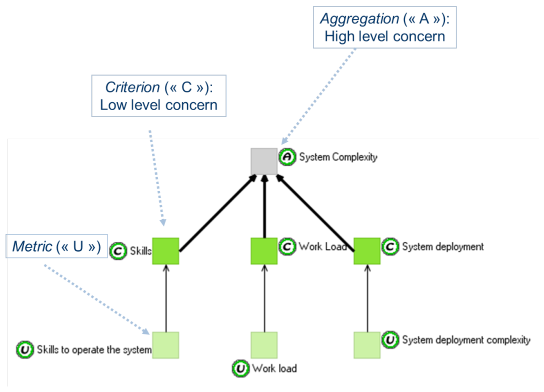

Aggregation (« A »): High level concern
Criterion (« C »): Low level concern
Metric (« U »)
System Complexity
Skills
Work Load
System deployment
Skills to operate the system
Work load
System deployment complexity

Fig. 1. The three main concepts.

is not simply ordinal since the scales coming from different criteria are combined trough arithmetic operators to allow compensation among criteria. The underlying scale is often a numerical scale, such as the [0,1] interval in which the value 0 is judged unacceptable, and value 1 is judged perfectly satisfactory relatively to the concern of the criterion. An interval scale is a scale in which the notion of difference makes sense. In the scale [0,1], going from utility 0.1 to 0.2 is equivalent in terms of satisfaction gain to going from 0.8 to 0.9. Secondly, **commensurability** between the different criteria is required. Commensurability means that a same evaluation on different criteria has the same meaning. For instance, evaluation 0.3 shall have the same interpretation whatever the criterion. Commensurability is very complex to obtain for ordinal scales, as the user needs to compare all elements of all metrics together. By contrast, interval scales can be made commensurate more easily. An interval scale has two degrees of freedom. In order to fix entirely an interval scale, it is sufficient to fix the utility for two values of the metric. These particular elements will have the same utility on the different criteria, and are called reference elements. A reference level is an abstract level for which one can identify a reference element on each metric which corresponds to the abstract level. In the literature, several reference levels have been defined.

Completely Satisfactory the criteria is completely met. It is a saturation level in the sense that one cannot do better than this level in terms of satisfaction.

Budget (target value) This is the expected value in the requirement provided by the customer.

Satisficing This word has been invented by the sociologist H. Simon. The decision maker is happy when this value is reached, even if better elements exist. The user does not basically look for better elements than the satisficing element. The satisficing element usually corresponds to the budget element.

Neutral This level is neither good nor bad. The decision maker is indifferent when he encounters such element. Values better than the neutral element are considered as Good whereas values worse than the neutral element are considered as Bad.

Not satisfied at all The criterion is not met at all for this value. This is also a saturation level as one cannot be worse than this level.

Unacceptable It is similar to “not satisfied at all”, except that one means that this is a veto value. An architecture having an unacceptable value on a criterion cannot be selected.

For simplicity in the following case the utility function is described by 3 levels of criteria satisfaction level: **Not satisfying at all value**, **Budget** and **Completely satisfactory** corresponding respectively to 0, 0.5 and 1 satisfaction marks as illustrated in Fig. 2.

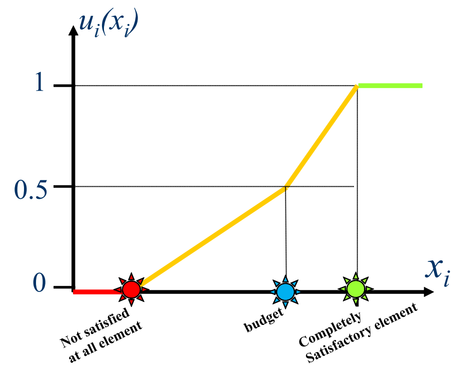


Fig. 2. Shape of a basic utility function

C. *MCDA model building*

The construction of a MCDA model is composed of the following stages [22]:

*1) **Stage 1**:* It is the structuring phase. The goal is to construct a tree representing a hierarchy of concerns using the basic concepts defined above in which the root represents the overall evaluation, and the leaves are the metrics. All nodes except the leaves to return a numerical evaluation that is a satisfaction degree. The process is decomposed as the following.

Specify concisely the expected issue of the global evaluation. The first step consists in defining in one sentence what the overall evaluation aims at representing. This helps to prevent from taking into account non relevant aspects. This also helps, within future discussions, to retarget the debate by reminding when necessary the objective of evaluation.

Identify the stakeholders. The objective of this step is to itemize the list of stakeholders being involved and in particular to identify all necessary competences. A significant number of stakeholders may intervene in the decision aiding process. Meaning stakeholders are, among others, the **Decision Maker(s)**, responsible of the decision that will be taken. It can be an end user or a customer. Other important stakeholders are the **Expert(s)** which can be any person that may give his opinion in order to help the construction of the evaluation model. Examples of experts are operational and technical experts. The customer may be seen as an expert in some situations. For complex decision problems involving many stakeholders a **Facilitator** may be useful. He is more likely an external consultant rather than a ”classic” stakeholder. He helps the stakeholders to structure the debate by bringing a methodology and his external understanding of the problem, by asking the relevant questions and reformulazing what the stakeholders express. He also helps to establish confidence amongst stakeholders and to converge towards a common understanding of the problem shared by all stakeholders.

Build a hierarchy of concerns. The number of relevant criteria is often relatively large and can be larger than several dozen in complex problems.With more than 7 or 8 criteria, psychological studies [19] have shown that human being generally use only simplistic strategies. He decomposes the criteria into two groups: the important ones and the other ones. Only the important criteria are dealt with in a subtle way, taking them separately. The human being makes only a global reasoning on the less important criteria - based for instance on some kind of simple average. The less important criteria are thus not taken into consideration in a subtle way.
For these reasons, one shall proceed at a structuring phase. The aim is to construct a hierarchy of concerns, that is, several nested levels of aggregations. In order to perform such a hierarchy, one shall succeed in grouping the criteria according to a classification that makes sense of the stakeholders. At the end of this step, one shall obtain the relevant criteria together with their organization in a tree. Top-down (objective oriented) and Bottom-up (alternative oriented) approaches are possible even if mixing both approaches is often preferable.

Operationalize the concerns. The concerns that have been identified and grouped in a tree in preceding step are abstractions. We need to identify measurable variables representing these abstractions at best. The difficulty is to identify the right measurable datum which synthesizes the different aspects included within the consequence of each studied concern. In practice, there are often many metrics that can be seen as good representative of a given concern. There are two types of metrics: the **natural metric** is typically a statistical indicator whose expression can be easily described. A particular case concerns the proxi-attributes. These are metrics which link on the concern is not at first sight obvious. To illustrate this, let us mention the metric "concentration of pollutant" to measure the consequence on the concern of the effect on health. The **constructed metric** is computed. The stakeholders shall wonder how to specify a given concern into a value that can be computed. This is not always simple, especially when the available information is in limited quantity.
There are two types of **constructed metrics**. The first type is the **aggregated metrics** constructed by aggregating the values of small components such as the "general load of a system combining the load of sub-systems". The other type of metric is **ex-nihilo metrics** for the purely subjective judgments such as the "workload of an operator operating a system".

Specification of the preference ordering on the metrics. Each criterion shall depict a different aspect of the overall evaluation. In any case, the larger the utility the better it is. A metric is not any variable having a more or less direct influence on the overall evaluation of the alternatives. There necessarily exists a preference regarding each criterion. This preference indicates the values of the metric that are judged good, fair, bad, … The preference associated to a metric regarding the consequences on a concern is specified through a preference relation called criterion. A criterion is characterized by a value function which returns a utility to each value of the metric. This is the utility function. This specifies the better and the worse. One shall specify the sense of the preference regarding a metric, all else being equal on the other metrics. The most commonly encountered senses of variation are "the larger the metric the better" as for a performance metric, "the smaller the metric the better" as for a cost metric and "the closer to a given value the better" as for some soft requirements.

Validate a family of criteria. This step validates the hierarchy of criteria. It consists in checking whether the selected criteria and metrics satisfy to some elementary properties. Some important conditions to consider are the set of selected metric form the only variables on which the overall evaluation will be based. All information necessary to assess the alternatives shall be contained in the list of metrics. The set of criteria shall be independent.
Multi-Criteria Decision Aid has formalized the concept of consistent family of criteria. A family of criteria is said to be consistent if the following three properties are fulfilled: **Exhaustiveness** - The family of criteria is sufficient to compare any alternativeness (i.e. there is no missing criterion). If two alternatives are judged identical regarding all the criteria (same preference), then these two alternatives shall be globally judged identically. **Cohesion** - No criterion is useless. For each criterion, there exist at least a couple of alternatives for which one is globally strictly preferred to the second one such that, if one is strictly preferred to the other on this criterion, they are judged identically on the other criteria. **Non-redundancy** - There are no two identical criteria, suppression of one criterion yields the violation of one of the two previous properties.

*2) **Stage 2:*** It consists in quantifying the evaluation tree on the criteria nodes. In other words, we need to construct a judgment for each attribute separately. This amounts to ask "Is this value for this attribute is good or bad?". This is quantified by a satisfaction function. The construction of this function results from an interview with the domain expert. It is characterized by some thresholds that need to be identified. For the construction of the curve on intermediate values of satisfaction, we use dedicated methods from measurement to quantify the preferences of the expert into numerical utilities. A utility function on a metric is a scale representing the preferences of a stakeholder. There are basically two types of scales: the ordinal scales and the interval scales. An ordinal scale is rather poor, and does not really permit to handle numbers, since usual arithmetic operations are not meaningful here. In an interval scale, the concept of difference makes sense.
Aggregating different criteria requires that they are commensurable. This implies that one shall be able to compare any element of a metric with any element of any other metric. It is not possible to ask to a stakeholder to perform such a comparison since the direct comparison of elements belonging to different metrics does not make sense for human beings. The

MACBETH approach[10] allows solving these difficulties in a way that is completely satisfactory for the stakeholders and is relevant from a mathematical standpoint.

MACBETH approach overview. To explain the MACBETH approach, let us first note that in an interval scale, the notion of comparison of discrepancies makes sense. Moreover, an interval scale is given up to a dilation and a shift. This implies that it is enough to fix two points in an interval scale to entirely fix the scale. This is done for the utility function (which corresponds to an interval scale) on each criterion. As a consequence, the commensurateness assumption will be satisfied if one is able to find on metrics two elements, having the same meaning throughout all criteria. The second ingredient necessary to construct an interval scale - a specification of the difference between pairs of elements of the metrics - in terms of satisfaction degrees.

Definition of the reference levels on each attributes. The value of the utility function is interpreted as a degree of satisfaction regarding the preferences of stakeholders. This satisfaction degree corresponds to the same scale on all attributes. Such a degree belongs to the [0,1] unit interval, where 0 corresponds to the total absence of satisfaction and 1 corresponds to the complete satisfaction. The two reference levels are identified to these two levels. The stakeholder is thus asked to identify on each metric two elements :

- The first, named ”U”, is thought by the stakeholder as completely unsatisfactory relatively to his concerns w.r.t. criterion, its satisfaction will be valuated to 0.
- The second, named ”P” is considered as perfectly satisfactory[10], its satisfaction will be valuated to 1.

Identification of referent values on each attribute. If the metric is continuous and is an interval, it is not possible to elicit the utility function for all values of the metric. In practice, we elicit the utility function from the preferences of the stakeholder for only a finite set of elements of the metric. The two reference levels U and P shall be part of the considered subset of the metric in order to normalize the construction of the utility over these elements. The utility function of the metric is constructed by linear interpolation from its value on the identified subset.

Let us discuss on the construction of the elements of the subset when the metric is continuous. Since the [0,1] unit interval is a bounded unipolar scale, and is thus bounded from above and below, the two utilities 0 and 1 correspond to saturation levels. In the example of Figure 3, any element lower than 5 (resp. greater than 13) has a utility equal to 0 (resp. 1). More generally, if the utility function is non-decreasing (i.e. the larger the value of the metric, the better), U is the largest element of the metric for which all the lower values are uniformly judged as unacceptable. Likewise, P is the smallest element of the metric for which all the above values are uniformly perfectly satisfying. The same identification of the relevant levels U and P can be made for non-increasing utility functions and other types of monotonicity.

We identify the two elements U and P as we have just described. These are the first two elements of subset of the

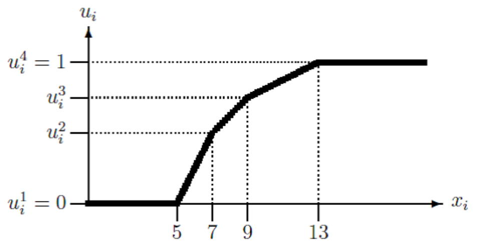


Fig. 3. Piecewise function caracterized by the values {5,7,9,13}

metric. Let us explain how to define the other elements of the subset. Firstly, the number of intermediate points between U and P basically corresponds to the complexity of the utility function. In general, it is between 2 (no intermediate point) and 7. For the identification of subset values, the segmentation is not necessarily uniform, depending if the shape is convex, concave or more or less linear. The idea is that the majority of points is placed where variations are the more important. For concave curves, we put more points close to U than to P. This is the case in the example of Figure 3. For convex curves, we put more points close to P than to U. The identification of the subset is easier when the metric is a finite set. In this case, the subset is often equal to the metric definition set.

Construction of the utilities on the elements of metric subset. The stakeholder is asked to answer to types of questions regarding the utilities of the elements of the metric subset: **Ordinal information** Given two elements of the metric subset, what is the preference between these two elements? The stakeholder has the choice between three answers: (1) the first is strictly preferred to the second, (2) the first and the second are judged indifferent, (3) the second is strictly preferred to the first. **Cardinal information:** Given two elements for which the second is strictly preferred to the first, what is the satisfaction gain (attractiveness) when going from the first to the second? The answer shall be given within the following finite scale: **Unknown, Very Weak, Weak, Moderate, Strong, Very Strong, and Extreme**. Hesitation between several values can be expressed as an interval of values in the previous finite scale.

In the Figure 3, 13 is strictly preferred to 9, which is strictly preferred to 7, which is strictly preferred to 5. Moreover the cardinal information is gathered in the matrix I.

	13	9	7	5
13	-	Weak	Moderate	Very Strong
9		-	Weak	Strong
7			-	Moderate
5				-

TABLE I
EXAMPLE OF A CARDINAL INFORMATION MATRIX.

*3) **Stage 3:*** It consists in quantifying the evaluation tree on each aggregation node. One needs to aggregate the partial evaluations to obtain higher level evaluations. Considering for instance an aggregation of three criteria, this amounts to know whether the satisfaction attached to an alternative that is for instance good over the first criterion, fair on the second criterion and bad on the last criterion, is considered as rather good, or rather bad. It is likely that the overall satisfaction will equal some value in between. A trade-off or compromise shall be made amongst all the criteria used to compute the aggregation node. This is obtained with the help of compensatory aggregation functions. People use here most often the weighted sum. At the end of this stage, the evaluation model is thoroughly specified.

We are interested here in the construction of the aggregation function at a node. Basically, there are four types of methods to learn the parameters of aggregation methods:

Direct assessment of the parameters. The stakeholders directly assign numerical values to the parameters of the decision model. This approach is not satisfactory. On one hand, the semantics of the weights is not so clearly understood by humans. The concept of weight is used in different methods (weighted sum, weighted ordered average, weighted minimum, weighted majority,...), and the weights have different meaning in each method. On the other hand, for a weighted sum, weights do not make sense in the criteria are not commensurate.

The notion of importance can be rigorously defined once commensurate scales are defined on the metrics. For instance, a criterion is twice as much as important as another criterion if an increase of one unit on the first criterion is equivalent to an increase of two units on the second criterion. This unit or standard corresponds to the commensurateness assumption. In order to identify the precise relative importance ratio between two criteria, one shall ask the stakeholder to identify tradeoffs between these two criteria. Thus, it is delicate to ask directly the weights to the stakeholders. A learning phase by indirect questioning is preferable by large.

Expression of the preferences as a language. In Artificial Intelligence, it is usual to describe all possible models as well-formed formulas defined from a language. This allows a compact representation of the preferences. The idea of these approaches is that the stakeholder can then directly express his preferences in this language. This approach is not possible with the model that we consider.

Elicitation of the parameters from learning examples provided by the stakeholder. A very commonly used approach is to learn the decision model parameters from a set of learning examples provided by the stakeholders. Such examples are typically comparisons of alternatives or assessment of alternatives, which values of all attributes or criteria are known. From the learning data, one can analyze the potential inconsistencies or the incompatibilities with the model that is considered and analyze the completeness of the learning data.The difficulty of this approach arises when the previous set of compatible parameters is large:it is not easy to determine the most relevant learning examples that shall be given in order to reduce the size of the set of compatible parameters as much as possible. This difficulty yields to the last approach.

Elicitation of the parameters from learning examples constructed by the approach. In this approach, the parameters of the decision model are also constructed from a set of learning examples. But instead of asking the stakeholders to provide them, the idea is to construct a set of alternatives from which questions will be asked to the stakeholders. These alternatives are optimally determined so as to maximize the accuracy of the identification of the model parameters. This has some links with some statistical methods such as experiment design, or active machine learning.

The corresponding elicitation process is explained hereafter.

Identification of the relevant alternatives: the binary alternatives. The preferential information that can consider here is a generalization of what is used in the MACBETH approach. The MACBETH approach is dedicated to a weighted sum. For a weighted sum, the weight of a criterion represents its sole importance in the aggregation. We wish to generalize this to a 2-additive Choquet integral. On top of representing the importance of criteria, it can also model interaction between pairs of criteria. Instead of allowing the options to be perfectly satisfactory on one attribute only, one may allow the options to be perfectly satisfactory on two attributes at the same time. These alternatives are called binary as they can take only two values on the different criteria.

Construction of the parameters from the binary alternatives. The stakeholder is asked to provide some ordinal and cardinal information on the binary alternative. The ordinal and cardinal information are of the same nature as presented before. More precisely, the stakeholders are asked to answer questions regarding the utilities of the elements of binary alternative. **Ordinal information:** Given two elements x and y of the binary alternative, what is the preference between these two elements? The stakeholder has the choice between three answers: (1) x is strictly preferred to y, (2) x and y are judged indifferent, (3) y is strictly preferred to x. **Cardinal information:** Given two elements x and y of the binary alternative for which x is strictly preferred to y, what is the satisfaction gain (attractiveness) when going from y to x? The answer shall be given within the following same finite scale than for the utility function with the same pratice when hesitation occurs.

The analysis of the inconsistencies is much more complex than in the utility construction. This is due to the monotony conditions for the 2-additive Choquet integral. The description of the handling of the inconsistency is not the purpose of this document. One can see that the number of elements of binary alternatives increases with the square of the criteria at the level.

IV. Industrial case-study

The design of complex systems such as radio communication products requires taking into account various and sometimes contradictory concerns such as security and performance. Indeed, radio communication equipment exhibits strong requirements in terms of size, weight, power consumption, security and real-time performance. One of the most challenging aspects in system engineering is to analyze the combination of numerous concerns.

A. Secure Radio Architecture.

A secure radio platform is basically divided into three parts: The **Red security domain** receives sensitive information from the user point of view (data plan) such as plain text data that need to be ciphered; The **Black security domain** deals with nonsensitive information that are ciphered for data information and may be ciphered or not for control information; An **Information security domain** (InfoSec) handles communications between Red and Black domains. It ciphers data information from Red to Black domain and deciphers them from Black to Red domain using cryptographic channels. Control information may go between Red and Black domains without ciphering using bypass channels. For strong security and safety needs, a physical separation is enforced for the Red, Black and InfoSec domains. Each domain is implemented by a dedicated board in the radio equipment and has its own independent processor. The introduction of multi-core processors, hypervisor and separation kernel technologies in embedded systems allows a new security/safety architecture with a logical separation between the Red, Black and InfoSec domains. Basically, each domain may be implemented on a single multi-core processor. Multiple processors may be replaced by a single multi-core processor at lower frequency. This reduces power consumption as it roughly grows linearly with the processor frequency and the number of processors.

B. Hardware/Software Architecture.[6]

A radio platform is the set of software and hardware layers that provide the services required by the Software Radio Protocol (SRP) application layer through Application Programming Interfaces (APIs). A radio platform includes system components: Radio Devices (RD) (e.g. Ethernet Device, Audio device) and others Services (e.g. management service, IP and routing service). The SRP application and Software-Defined Radio (SDR) platform components may be designed for different security/safety levels (e.g. Common Criteria (CC) for security and/or DO178 for safety). Figure 4 presents the SRP application high level architecture.

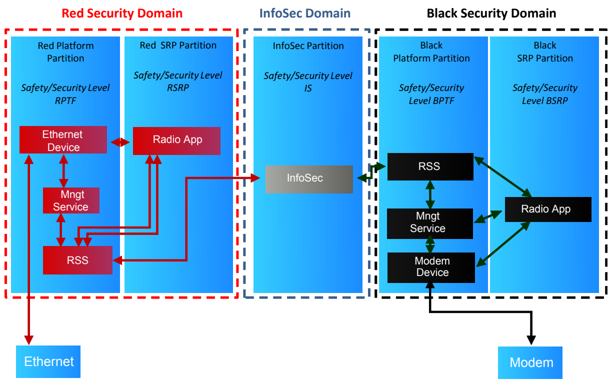


Fig. 4. SRP application high-level architecture

In addition to the SRP application components (Red and Black Radio App), the use case architecture consists of the following SDR platform components: **the Ethernet Device** abstracts an Ethernet network interface of the target SDR platform, **the Management Service** checks and dispatches control and management requests to SRP and platform components. For instance, it allows the configuration of component properties such as the MAC address or the transmission power of the radio equipment, **the Radio Security Service** (RSS) provides security channels to cipher/decipher user information, and forward control information without encryption (bypass), **the Modem Device** abstracts the Physical layer implemented on DSP, FPGA and the Radio Frequency (RF) front-end(s).

C. The experiment.

This experiment focus evaluations on the SRP sub-system of the SDR. Based on the same logical model, different deployments of the component-based radio application may be compared according to various criteria. Each security/safety partition may be physically isolated in boards or logically isolated in virtual machines (VMs). As described above there are three security domains: Red, InfoSec and Black domains, and five security/safety partitions for Red Application components, Red Platform components, InfoSec components, Black Application components and Black Platform components. Each domain and partition may have different Saftey/security level depending on the final product target application. Considering the number of boards and VMs reliable variants have been automatically derived from a variability model connected to business architecture patterns described in [11]. These reliable variants are presented in the Table II.

Solution	Description	Board(s)	VM(s)
S1	1 board per safety partition	5	0
S2	1 board per security domain	3	0
S3	1 board per sec. dom. and 1 VM per saf. part.	3	5
S4	1 board and 1 VM per security domain	1	3
S5	1 board and 1 VM per safety partition	1	5

TABLE II
CANDIDATE ARCHITECTURE VARIANTS.

D. Evaluation Criteria.

A solution may be applicable to a specific usage in the SDR productline. The objective is selecting the best design for a hand-held SDR (low power, safe and secure, high availability). The identified criteria are listed in the Table III.

Id	Criteria	Not satisfying at all value	Budget value	Completely satisfactory value
1	Software part. for Security	0	3	3
2	Hardware part. for Security	1	3	3
3	Software part. for Safety	0	3	3
4	Hardware part. for Safety	1	3	3
5	Used CPU Resource ratio	0.55	0.50	0.25
6	SoC Lifetime (h)	50000	60000	90000
7	Communication overhead (us)	1500	1000	200
8	Maintenance period (h)	90000	70000	60000
9	Cost per equipment (Euro)	1000	600	200
10	Power consumption (mWh)	2000	1700	1000

TABLE III
EVALUATION CRITERIA.

The Criteria are aggregated into several Aggregations mapping the principal concerns of the architecture description. Such mapping is often questionable and requires experts consensus. The resulting Aggregations are the following:

- **Security** aggregates Criteria 1 and 2;
- **Safety** aggregates Criteria 3 and 4;
- **Availability** aggregates Criteria 5 and 6;
- **RoI** aggregated Criteria 8 and 9;
- Criteria 7 and 10 remain untouched.

Aggregations and Criteria are aggregated under a single Aggregation, the model root, representing the overall assessment. The resulting model overview is represented in Figure 5.

Experts expressed the constraint that a solution presenting a top-level Aggregation or Criterion evaluated as "Not satisfying at all" must be evaluated under "Budget". This constraint has been translated by automated learning into a nearly global complementarity between these Aggregations and Criteria weights; the complementarity between criteria is interpreted as the min of evaluation of theses criteria by the 2-additive Choquet integral. The resulting relative weight schema is represented in the Figure 6.

V. Results and conclusion

For concrete understanding of the proposed approach value we compare results of the proposed approach with the usual weighted sum approach, both are using the same aggregation model and utility functions. The expressiveness of weighted sum make us adapt manually the relative weight of top-level Aggregations or Criteria as described in the Figure 7.

Results are compiled in the Table IV[2]. The comparison criteria is the overall score interpreted as a utility function, the higher is the better: a 0 score is interpreted as "Not satisfactory at all", a 0.5 score is interpreted as "Budget" and a 1 score is interpreted a "Completely satisfactory". The objective is then selecting solutions evaluated above 0.5 and, in case of multiple selection, choose the one with the highest mark.

Solution	Weighted sum @1000Mhz	Myriad @1000Mhz	Weighted sum @800Mhz	Myriad @800Mhz
S1	0.41	0.24	0.43	0.29
S2	0.54	0.36	0.56	0.43
S3	**0.62**	0.32	0.64	0.39
S4	0.59	**0.42**	**0.74**	**0.78**
S5	0.56	0.39	0.72	0.76

TABLE IV
Overall evaluation results.

A. Results

At 1000 Mhz Weighted sum proposes four satisfying solutions meanwhile Myriad-based evaluation cannot find any satisfying. Looking into details, Power consumption is "not satisfying at all" in all cases. Good scores on other criteria are compensating Power. The Myriad-based evaluation make us conclude there is **no satisfying solution**. This conclusion is operationally valid for the target usage. Because Power is outside acceptable range while CPU Resource exceeds the expectations, lowering SoC frequency may improve Power evaluation while keeping CPU Resource satisfying. At 800 Mhz Weighted sum evaluation proposes four satisfying solutions. Other criteria scores balance the unacceptable score of Power for S2 and S3 meanwhile S4 and S5 are identified as satisfying solutions. Myriad-based evaluation exclude all solutions other than S4 and S5. These solutions are operationally acceptable, both methods agree on this.

[2]Full evaluation models and results are available on demand.

B. Comparison justification

When deciding, one has to justify the choice. Myriad generates an argumentation report justifying evaluation and comparison towards the evaluation model. As example, the following is the raw result generated from the Myriad evaluations comparison at 800 Mhz, focusing on S4.

S4 is clearly preferred to S1 on the criterion "SDR Overall Assessment". **"S4" is preferred to "S1"** since the intensity of preference of "S4" over "S1" on the criteria "Communication overhead", "RoI" and "Power" is **MUCH LARGER** than the intensity of preference of "S1" over "S4" on criterion "Availability".

S4 is a bit preferred to S2 on the criterion "SDR Overall Assessment". **"S4" is preferred to "S2"** since **the large importance of the criteria "RoI" and "Power" reinforces** the relative strength of "S4" compared to "S2" on these criteria, **the small importance of criterion "Availability" minimizes** the relative strength of "S4" compared to "S2" on this criterion.

S4 is preferred to S3 on the criterion "SDR Overall Assessment". **"S4" is preferred to "S3"** since the intensity of preference of "S4" over "S3" on the criteria "Communication overhead", "RoI" and "Power" is **MUCH LARGER** than the intensity of preference of "S3" over "S4" on criterion "Availability".

S4 is almost similar to S5 on the criterion "SDR Overall Assessment". **"S4" is preferred to "S5"** since the intensity of preference of "S4" over "S5" on the criteria "Communication overhead" and "RoI" is **MUCH LARGER** than the intensity of preference of "S5" over "S4" on nothing.

Despite its automatic syntax, the generated argumentation helps in producing justification report, producing a complete argumentation of the evaluation for each Aggregation.

C. Conclusion

In this paper we illustrated the use of a tooled method for comparing evaluations of different solutions to a given problem in an objective way. Evaluating is always a irrational activity. The preference model synthetizes experts know-how. It is built by a method attempting to bring maximum rationality: utility and weights are computed by automated learning based on experts decisions. An often used approach consists in changing the criteria weights during the decision process in order to reach the expected alternative. This is of course very debatable. The methodology proposed represents better the decision maker preferences than when fixing directly utilities and weights. A preference model addresses a mean of answering to a question, not the definition of truth. If the radio was designed for being embedded in a vehicle, the preference model, although having the same aggregation structure, would have different utilities and weight for criteria such as Power consumption or CPU resource usage. The results would have been different.

The presented tooled method gives an evaluation thanks to a preference model, with evaluation result justifications. Deciding implies responsibility and for this reason remains

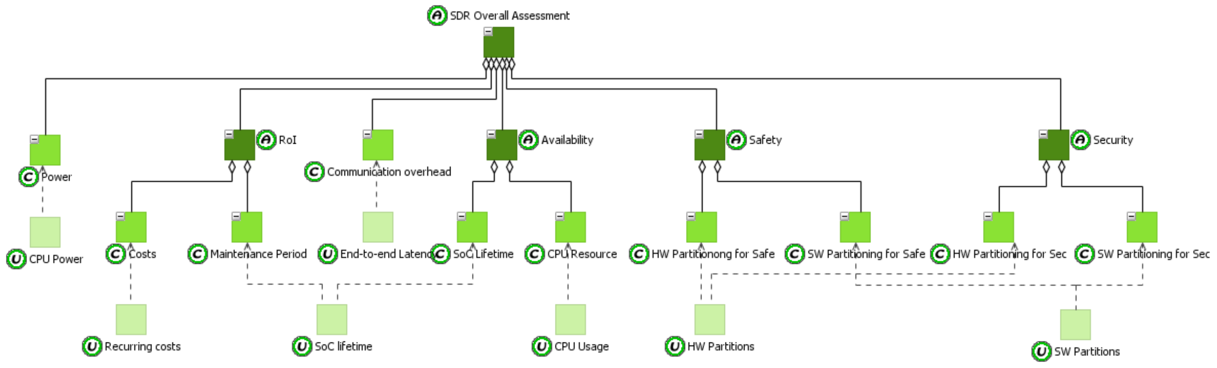


Fig. 5. MCDA model

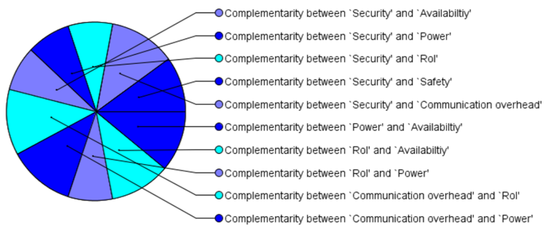


Fig. 6. Aggregation and Criteria relative weights, MYRIAD method.

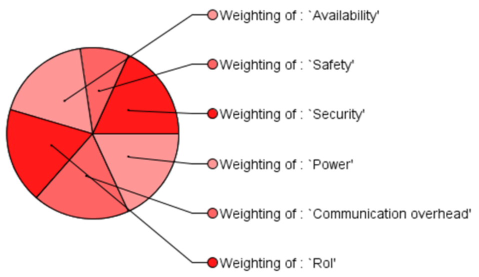


Fig. 7. Aggregation and Criteria relative weights, Weighted sum method.

an expert activity. The role of tooling is only to help decision maker in his task.

According to the experiment both evaluation methods highlight the best solutions. The Weighted sum proposes false positives because it requires independent variables. This hypothesis is not true in our case: we need to model "if one of Criteria is Not Satisfying At All then Evaluation is under Budget". Giving artificially strong weights to such criteria does not work here because of compensation. The Weighted sum is not adapted to decision making problems such as choosing a design for a given usage because one of application hypothesis is not satisfied. Myriad uses the Choquet integral for aggregating criteria satisfaction. It acts as a Weighted sum when variables are independent and manages variables interaction. It is an adapted tool to the problem class illustrated.

The presented tooled method requires the capability of sorting criteria, not possible when the criteria number is high and/or preference sorting is not possible. In this case one needs a new preference model and aggregation function class such as Generalized Additive (GAI) model[18].

REFERENCES

[1] http://www.esteco.com
[2] http://www.simul8.com
[3] http://visadecisions.com/
[4] http://www-05.ibm.com/de/events/embedded-world/WP-Smarter-system-development.pdf
[5] Defense acquisition guidebook. https://dag.dau.mil/
[6] International Radio Security Services API Specification. WINNF-09-S-0011. V2.0.0
[7] Manual for the operation of the joint capabilities integration and development system
[8] Biltgen, P., Ender, T., Cole, B., Mavris, D., Biltgen, P., Mavris, D., Molter, G.: Development of a collaborative capability-based tradeoff environment for complex system architectures. (2006)
[9] Biltgen, P., Mavris, D., Molter, G.: A methodology for technology evaluation and capability tradeoff for complex system architectures
[10] Bana e Costa, C., Vansnick, J.: A theoretical framework for Measuring Attractiveness by a Categorical Based Evaluation TecHnique (MACBETH). (1994)
[11] Degueule, T., Filho, J.F., Barais, O., Acher, M., Noir, J.L., Madelénat, S., Gailliard, G., Burlot, G., Constant, O.: Tooling support for variability and architectural patterns in systems engineering (2015)
[12] Gerd Gigerenzer, W.G.: Heuristic decision making. Annual Review of Psychology 62, 451–482 (2011)
[13] Kazman, R.; Abowd, G.B.L.W.M.: Saam: A method for analyzing the properties of software architectures
[14] Kazman, R.; Abowd, G.B.L.W.M.: Software Reliability Theory. Encyclopedia of Software Engineering (1994)
[15] Kazman, R.; Klein, M.C.P.: Atam: Method for architecture software engineering institute (2000), http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html
[16] Klein, M., Ralya, T., Pollak, B., Obenza, R., Gonzales Harbour, M.: A Practitioner's Handbook for Real-Time Analysis (1993)
[17] Labreuche, C., Le Huédé, F.: Myriad: a tool suite for MCDA pp. 204–209 (September 7-9 2005)
[18] Labreuche, C., Grabisch, M.: A comparison of the GAI model and the choquet integral w.r.t. a k-ary capacity pp. 54–65 (2015), http://dx.doi.org/10.1007/978-3-319-23240-9_5
[19] Miller, G.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. The Psychological Review 63, 81–97 (1956)
[20] Montmain, J., Labreuche, C., Imoussaten, A., Trousset, F.: Multi-criteria improvement of complex systems. Information Sciences 291, 61–84 (2015)
[21] Moody, K.: Issues and recommendations for instituting capability-based planning in the canadian forces (2005), http://www.cfc.forces.gc.ca/papers/csc/csc31/mds/moody.doc
[22] Roy, B.: Decision aiding today: what should we expect? (1999)

Session 10

Network & Simulation

Thursday 28th, 09:00 – Guillaumet

Timing verification of real-time automotive Ethernet networks: what can we expect from simulation?

Nicolas Navet, University of Luxembourg
Jan R. Seyler[1], Streyler GbR, Germany
Jörn Migge, RealTime-at-Work, France

Abstract: Switched Ethernet is a technology that is profoundly reshaping automotive communication architectures as it did in other application domains such as avionics with the use of AFDX backbones. Early stage timing verification of critical embedded networks typically relies on simulation and worst-case schedulability analysis. When the modeling power of schedulability analysis is not sufficient, there are typically two options: either make pessimistic assumptions or ignore what cannot be modeled. Both options are unsatisfactory because they are either inefficient in terms of resource usage or potentially unsafe. To overcome those issues, we believe it is a good practice to use simulation models, which can be more realistic, along with schedulability analysis. The two basic questions that we aim to study here is what can we expect from simulation, and how to use it properly? This empirical study explores these questions on realistic case-studies and provides methodological guidelines for the use of simulation in the design of switched Ethernet networks. A broader objective of the study is to compare the outcomes of schedulability analyses and simulation, and conclude about the scope of usability of simulation in the design of critical Ethernet networks.

Keywords: timing verification, timing-accurate simulation, ergodicity, automotive Ethernet, simulation methodology, worst-case response time analysis.

1 Context and objectives of the study

Ethernet is meant in vehicles not only for the support of infotainment applications but also to transmit time-sensitive data used for the real-time control of the vehicle and ADAS functions. In such use-cases, the temporal behavior of the communication architecture must be carefully validated. Early stage timing verification of critical embedded networks typically relies on simulation and worst-case schedulability analysis, which basically consists in building a mathematical model of the worst possible situations that can be encountered at run-time.

When the modeling capabilities of schedulability analysis is not sufficient, which given the complexity of today's architectures is in our experience in many practical situations the case (see [Na13,Na14] and §2.4), there are typically two possibilities. The first option is to make pessimistic assumptions (e.g., modeling aperiodic frames as periodic ones), which is not always possible because for instance it may result in overloaded resources (e.g., link utilization larger than 100%). The second option is to ignore what cannot be modeled (e.g., ignoring transmission errors, aperiodic traffic, etc). Both options are unsatisfactory because they are either inefficient in terms of resource usage or potentially unsafe. In addition, it can happen that schedulability analysis tools provide wrong results, most often because the analysis' assumptions are not met by the actual implementation, or possibly because of numerical issues in the implementation (e.g., if floating point arithmetic is used), or simply because the analysis is flawed (see for instance [Da07]).

To overcome these issues, we believe that it is needed to use simulation along with schedulability analysis, so that the results of the two techniques can be cross-validated. Compared to schedulability analysis models, simulation models can be more realistic since it is feasible for a network simulator to capture all timing-relevant characteristics of the communication architecture and reproduce complex traffic patterns specific to

- A higher-level protocol such as SOME/IP SD [Sey15], or the many different frame triggering conditions in AUTOSAR Socket Adapter (see [SoAd] §7.2.2),
- An applicative-level software component.

The main shortcoming of simulation is that it does not provide any guarantees on the relevance of the results, and the user remains always unsure about the extent to which simulation results can be trusted.

[1] Jan Seyler was at Daimler AG, Mercedes-Benz Cars Development, at the time the study was conducted. An oral-only presentation with the same title was given at SAE World Congress 2015, "Safety-Critical Systems" Session, Detroit, USA, April 21-23, 2015.

Simulation can lead to wrong decisions because of mistakes in methodology (e.g, simulation time, number of experiments, etc) or simply because the performance metrics under study are just out-of-reach of simulation. The two basic questions that we aim to study here is what can we expect from simulation, and how to use it properly? This empirical study explores these questions and provides methodological guidelines for the use of simulation in the design of switched Ethernet networks. A broader objective of the study is to compare the outcomes of schedulability analyses and simulation, and conclude about the scope of usability of simulation in the design of critical Ethernet networks.

We paid a special attention in this study that the models used in simulation and schedulability analysis are in line, which means that they model the same characteristics of the system and make the same set of simplifying assumptions (see §2.1) regarding behaviors of the system that we believe are not central in this study. In many practical cases, this will however not be the case because the schedulability analyses available today are not able to capture the whole complexity of most communication architectures.

The article is organized as follows. We first study the following methodological questions:

- Q1: is a single simulation run enough or should the statistics be made out of several simulations with different initial conditions since simulation results depend on the initial conditions?
- Q2: can we run several simulations in parallel and aggregate the results?
- Q3: what is the appropriate minimum simulation length?

Answering these three questions first requires to know whether the simulated system is *ergodic* (see §3.1) or not. We then assess the scope of usability of simulation by comparison with schedulability analysis, and explore the followings questions:

- Q4: are the latency upper-bounds derived by schedulability analysis, based on the state of the art of the Network Calculus, as used in this study, accurate wrt to the latencies that can actually occur in the worst-case?
- Q5: is simulation an appropriate technique to derive the worst-case communication latencies?

2 Experimental setup

2.1 System under study and assumptions

In this work, we consider a standard switched Ethernet network supporting uni- and multicast communication between a set of software components distributed on a number of stations. In the following, the terms *flow* or *streams* refer to the data sent to a certain receiver of a multicast connection; all *packets*, also called *frames*, of the same traffic flow are delivered over the same path.

In order to identify the primary impacting factors, the following set of assumptions is placed:

- The exact architecture of the communication stacks is not considered (e.g, AUTOSAR communication stack). It is assumed that frames are waiting for transmission in a queue sorted by frame priorities then arrival times. If packets have no priority, as in case-study #2, the waiting queue is FIFO,
- The routing of the packets to the destination nodes is static,
- It is assumed that there are no transmission errors,
- Nodes' clocks are drifting away with the clock drifts being random but constant over time (see §2.6). The clock drift rates used in the experiments (±200ppm and ±400ppm) are realistic in the automotive domain [Na12],
- There are no buffer overflows in the Ethernet switches which would cause packets to be lost. In practice, this has to be avoided and can be ascertained by schedulability analysis, or, with a high confidence, by simulation,
- The packet switching delays in the Ethernet communication switches is assumed to be upper bounded, and vary from packet to packet according to a uniform distribution in the interval [0.1* bound, bound],
- Streams of frames are periodic and the successive frames of a stream are all of the same size,
- The communication switches are all store-and-forward switches.

2.2 Case-studies

The 3 case-studies described hereafter are considered in the experimentations. The first case-study is a prototype automotive network developed by Daimler [Se13, Se15]. The characteristics of tomorrow's large automotive Ethernet network, for instance Ethernet as a high-speed backbone supporting mixed-criticality traffic, are still unsure at the time of writing and we had no such large network at our disposal for the experiments. To perform experiments also with larger configurations, we included in this study two avionics configurations.

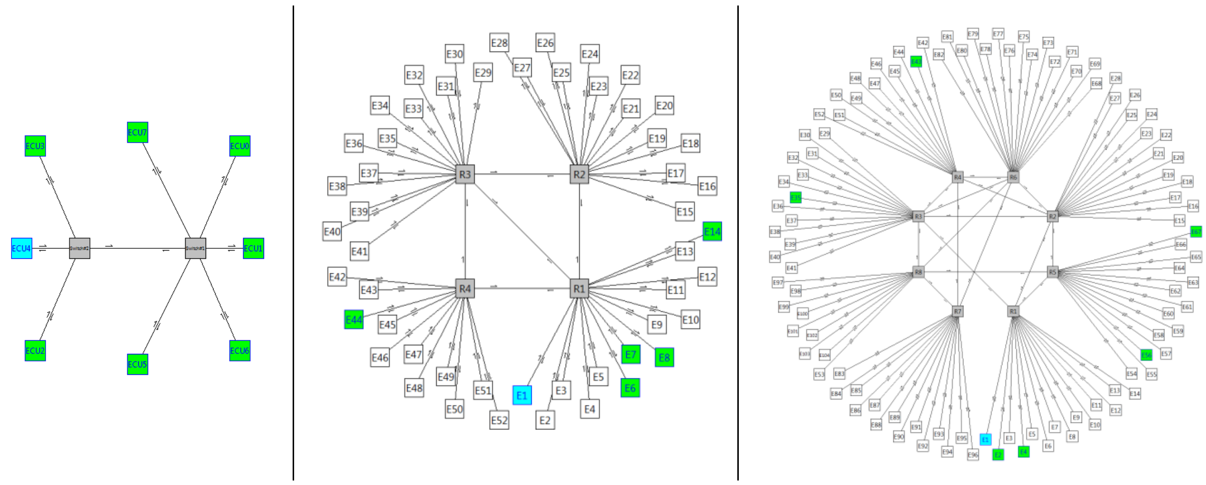


Figure 1: topology of case-study #1 (Daimler prototype network), case-study #2 (medium AFDX network) and case-study #3 (large AFDX network). A multi-cast stream is shown on each topology.

Case-study#1: Daimler prototype Ethernet networks. The use-case of this prototype network from Mercedes Cars is to support ADAS functions and exchange real-time control data. The exact specification of the case-study, such as the set of streams and the functions involved cannot be communicated for confidentiality reasons. Case-study #1, like it is done for the two other case-studies, is defined by a set of characteristics summarized in Figure 2 while its topology is shown in Figure 1.

Case-study #2: medium AFDX network. The second case-study is a sample configuration of RTaW-Pegase that is available upon request. It is scaled-downed version of the third case-study, which models the kinds of large AFDX networks that can be found in large civil aircrafts. In addition to the size, another difference with the two other case-studies is that the frames do not have priorities, they are thus scheduled on a FIFO basis in the nodes as well as in the transmission switches.

Case-study #3: large AFDX network. The third test configuration is a sample file of RTaW-Pegase that is available upon request. It aims to model the AFDX backbone networks [It07,Bo12] used in large civil aircrafts.

	Case-study #1	Case-study #2	Case-study #3
#Nodes	8	52	104
#Switches	2	4	8
#Switching delay	6us	7us	7us
#streams	58	3214	5701
#priority levels	2	None	5
Cumulated workload	0,33Gbit/s	0.49Gbit/s	0.97Gbit/s
Link data rates	100Mbit/s and 1Gbit/s (2 links)	100Mbit/s	100Mbit/s
Latency constraints	confidential	2 to 30ms	1 to 30ms
Number of receivers	1 to 7 (avg: 2.1)	1 to 42 (avg: 7.1)	1 to 83 (avg: 6.2)
Packet period	0.1 to 320ms	2 to 128ms	2 to 128ms
Frame size	51 to 1450bytes	100 to 1500bytes	100 to 1500bytes

Figure 2: Summary of the case-studies characteristics.

Due to space constraints, the results are not always shown in this paper for all the configurations. The reader is referred to [Na15] for the complete set of experimental results.

2.3 Software Toolset and performance evaluation techniques

This study has been conducted using RTaW-Pegase 2.1.7 timing analysis software, a product of RealTime-at-Work developed in partnership with ONERA research lab. RTaW-Pegase provides:

- *Timing-accurate simulation*. Conceptually, at each step *n* of the simulation, the system is fully characterized by a state *Sn* and the set of rules to change from state *n* to *n+1*: *Sn+1 = F(Sn+1)* is defined by the simulation model. The evolution of the system depends on this set of rules and the sequence of values provided by the random generator.
- *Worst-case latencies* (*i.e.*, worst-case response times calculation) using a state-of-the-art network calculus implementation [Bo11]. The pessimism of this schedulability analysis is known to be limited as it has been experimentally evidenced in the non-prioritized case in [Bo12] and in the experiments of §4.1,
- *Lower-bound on the worst-case latencies*. This information is key to estimate how tight the schedulability analysis is. The algorithm implemented in RTaW-Pegase is based on [Ba10].

Simulation results are of course much more fine-grained since the distributions of all quantities of interest can be collected during simulation runs. In the experiments of this study, the simulator is able to compute about 4.1 mega events per second on a single core of a standard desktop workstation (Intel I7-2600K 3.4Ghz), which means for instance that it can simulate 24 hours of communication for the first case-study in about 1h57mn, or less than 15mn with 8 simulations executed in parallel on a 8 core machines. This speed of execution is achieved by abstracting away all characteristics of the system without impact on its timing behavior. Speed is indeed crucial for simulation used in the design of critical systems since the samples of values collected must be sufficiently large to derive robust statistics with respect to the criticality of the application (i.e., samples sufficiently large for $1\text{-}10^{-5}$ quantile values, see [Na14]). Schedulability analysis is much faster that simulation, it takes about 15 seconds for the largest case-studies on the workstation used in the experiments. This speed of execution can be explained firstly because Network Calculus scales extremely well due to its low algorithmic complexity, and also because the implementation has been optimized since it has been started to be developed in 2009 in the Pegase collaborative project, see [Bo11].

2.4 Why schedulability analysis alone is not sufficient

Worst-case response time (WCRT) analysis, also referred to as schedulability analysis, is often considered as the technique that is the best suited to provide the guarantees that are needed in critical networks. Indeed, as soon as the workload submitted is bounded and the resource behaves in a deterministic manner, then it is always possible in theory to derive a worst-case schedulability analysis. Our experience with schedulability analyses has been however that they suffer from limitations in many practical cases due to the following issues:

1. *Pessimism* due to coarse-grained or conservative models (e.g., as in [Da12]) potentially leading to hardware resource over-provisioning. This might even rule out the use of analytic techniques in contexts where resource usage optimization is an industrial requirement,
2. *Complexity* that makes them error prone and hard to validate, especially since the analytic models used are most often not published[2] and the software implementing them is a black-box for the user,
3. The *inability to capture today's complex software and hardware architectures*. Using an inaccurate model can lead to inefficient resource usage or even unsafe design choices. What makes this perhaps the biggest issue is that it is hardly possible to foresee the effect of simplifying assumptions, given the non-monotonous and non-linear behavior of the model outputs.

An illustration of the latter point is that at the time of writing there is, as far as we know, no schedulability analysis that captures the complex frame transmission policies in the AUTOSAR Socket Adapter behavior [SaAd15], while simulation of this component is readily available in RTaW-Pegase for instance. Here we do not mean that schedulability analysis is never an appropriate technique, but simply that it is best suited to systems which have designed and implemented with simplicity, determinism and analyzability as primary design objectives. The reader can refer to [Na13, Na14] for a more thorough discussion on the complementarities of verification techniques in the design of automotive communication architectures.

[2] The core timing analysis algorithms of RTaW-Pegase have been published, e.g. [Bo07,Bo11], and partly formally proven in [Ma13,Bo14b].

2.5 Randomness in the simulation

Our simulation model of the Ethernet communication system is stochastic in the sense that two different simulation runs of the same configuration will not lead to the exact same trajectory of the system. Under the assumptions made in this study (e.g., no transmission errors, fixed packet size and period) the randomness comes entirely from:

- the offsets of the nodes, which is the initial time (wrt the network's origin of time) at which the nodes start to send messages (e.g., not all nodes will start to transmit simultaneously because of different boot times),
- the clock drifts of the nodes: the clocks that drive all activities on their host processor including the communication, do not operate at the exact same frequency,
- the switch commutation delay, that is the time it takes to copy a frame from its input port to a waiting queue on the output port.

These characteristics of the system are drawn at random according to the parameter ranges specified by the user (e.g, ±200 ppm maximum for the clock drifts), and their exact value depends on the seed of the random generator that is used for the simulation.

2.6 Modeling clock-drifts

The clocks of the CPUs of the network nodes never operate exactly at the same rate and thus they are slowly drifting away. These clock drifts result from various factors, the main ones being fabrication tolerance, aging and temperature (see [Mo12] for a discussion of the main factors of clock drifts and their quantification in automotive systems). Clock drifts are measured in "parts per million" or ppm, which expresses how slower or faster a clock is, as compared to a "perfect" clock. For instance, 1ppm corresponds to a deviation of 1µs every second. In this study, we assume that clocks drifts are constant throughout the simulation run and use the same model as in [Mo12]. For a given clock c driving an Ethernet node, its local time tc with respect to a global time t is determined as follows in the simulation model : $tc\ (t) = \varphi c + \delta c \cdot t$ where φc is the initial start time (the offset) of the node with regard to the bus time referential, and δc is the constant drift value. For instance, a drift rate of +100ppm means that $\delta c = 1.0001$. In this work, every node j is assigned a clock defined by the tuple $(\varphi j\ ,\ \delta j)$ which is chosen at random according the simulation parameters.

2.7 Performance metrics for frame latencies

The main performance metric for real-time communication networks is the communication latency, also called frame response time, which is the time from the production of a message until the reception by the stations that consume the message. The latency constraint, or deadline constraint, is the maximum allowed value for the response time. This deadline is typically inherited from applicative level constraints or regulatory constraints (e.g., time to answer a diagnosis request).

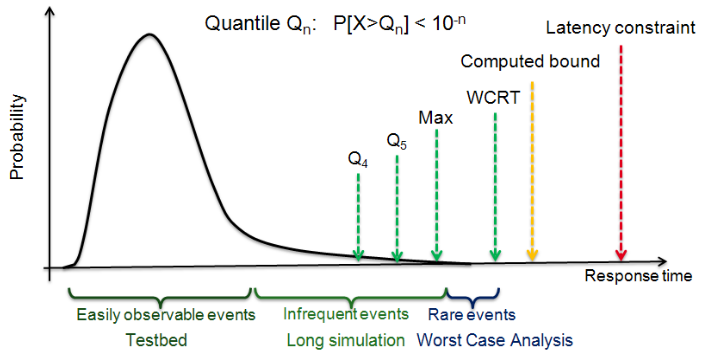


Figure 3: Metrics of the frame latencies and techniques to verify them. The black curve shows an idealized distribution of a frame response times (from [Na14]).

The aim of timing verification is to make sure that deadline constraints are met. Timing verification on models, by simulation or schedulability analysis, allows deriving a number of metrics on the frame response times. Those metrics, along with the corresponding timing verification techniques are shown in Figure 3. What is said in this paragraph holds equally for other quantities of interest such as buffer usage in communication switches and communications stacks.

The bound on the response time, which is the outcome of a schedulability analysis, is usually larger than the true worst-case possible response time (denoted by WCRT). In general schedulability analysis is pessimistic to an extent that cannot be predicted. However, in some cases it is possible to derive lower-bounds on the WCRT based on a pessimistic trajectory of the system that we know can happen. This is an analysis performed in §4.1. The maximum value seen during a simulation is most often less than the WCRT, here again the distance between both values is unknown and depends on the network configuration as shown in the experiments of §4.2. In the context of networks, the WCRT is also sometimes referred to as Worst-Case Traversal Time (WCTT), this is the term used in the rest of this document.

In the design phase, the quantiles of the quantities of interest are often other meaningful performance metrics. Formally, for a random variable X, a p-quantile is the smallest value x such that P[X>x] < 1- p. In other words, it is a threshold L such that for any response time,

- the probability to be smaller than L is larger than p,
- the probability to be larger than L is smaller than 1 – p.

For example, the probability that a response-time is larger than the (1-10^{-3})-quantile, denoted here by Q3 quantile or Q3 for short, is lower than 10^{-3}. For a frame with a period of 10ms, the Q3 will be exceeded on average once every $10^3 \cdot 10\text{ms} = 10^4\text{ms}$, that is 10s. Table 1 shows how quantiles translate to deadline miss frequency and average time between deadline misses, for frames with a period equal to 10ms and 500ms and deadlines assumed to be equal to quantiles.

Quantile	Deadline miss every	Mean time to deadline miss if period is 10ms	Mean time to deadline miss if period is 500ms
Q3	1000	10 s	8mn 20s
Q4	10 000	1mn 40s	≈ 1h 23mn
Q5	100 000	≈ 17mn	≈ 13h 53mn
Q6	1000 000	≈ 2h 46mn	≈ 5d 19h

Table 1: Quantiles and corresponding frame deadline miss frequencies for frame periods equal to 10ms and 500ms, and frame deadlines assumed to be equal to quantiles values (from [Na14]).

As exemplified in [Na14], verifying timing constraints with quantiles involves the following steps:

1. Identify the deadline for each frame,
2. With respect to the deadline miss probability that can be tolerated by the application, set the target quantile for each frame,
3. The objective is met if the target quantile value derived by simulation is below the frame deadline.

3 Methodology and parameters for simulation

In this section we explore the following questions pertaining to the choice of a proper methodology and setup for simulation:

- Q1: is a single simulation run enough or should the statistics be made out of several simulations with different initial conditions since simulation results depend on the initial conditions?
- Q2: can we run several simulations in parallel and aggregate the results?
- Q3: what is the appropriate minimum simulation length?

Answering these three questions requires first to know whether the simulated system is *ergodic*.

In the simulations performed in this work, except if otherwise stated, the following set of parameters was used:

- The clock drift of each node is chosen at random in ±200ppm. Simulations performed with ±400ppm returned results that were not significantly different,
- The offsets of the nodes are chosen at random in [0,100ms]. Simulations performed with offsets in [0,1s] returned results that were not significantly different,
- Each experiment is repeated 10 times with random offsets and clock drifts,
- Simulation time was at least 2 days of functioning time, corresponding to samples with more than 20 values above Q5 for sub-90ms flows.

3.1 Ergodicity of a dynamic process and practical implications

Intuitively, a dynamic system is said to be *ergodic* if, after a certain time, every trajectory of the system leads the same distribution of the state of the system, called the equilibrium state. If the system that is simulated is ergodic, it means that all statistical information can be derived from one sufficiently long simulation, since all simulations cover the state space of the system in a “similar” manner.

A single simulation of an ergodic system, or a few shorter simulations executed in parallel on a multicore machine, will lead to the same results as a large number of simulations with different initial conditions. This means from a practical point that we do not have to care about the number of distinct experiments that are to be performed, as long as each of them are “sufficiently” long, and the results obtained hold whatever the exact initial conditions of the system are.

The question that is experimentally investigated next is whether the ergodic property holds true or not for the system under study. In the latter case, this would imply that we would need to examine a large number of trajectories of the system, as done in the analytic techniques to calculate frame response time distribution in AFDX [Mau13] and CAN [Ze09, Ze10].

3.2 Do initial conditions have an impact on simulation’s results?

If the distributions of the quantities that are observed during the simulation are not identical for different initial conditions, then it implies that the simulated process is not ergodic. To empirically study that question, we performed for each case-study at least 10 simulations with different initial conditions:

- Random offsets and random clock drifts,
- Random offsets and fixed clock drifts,
- Fixed offsets and random clock drifts.

We are here interested in the frame latency distribution, our main performance metrics. We checked manually the convergence of the latency distributions obtained in different simulations for several frames in each case-study. The convergence could always be visually confirmed. This is for instance what is shown in Figure 4 for a 100ms frame of the first case-study.

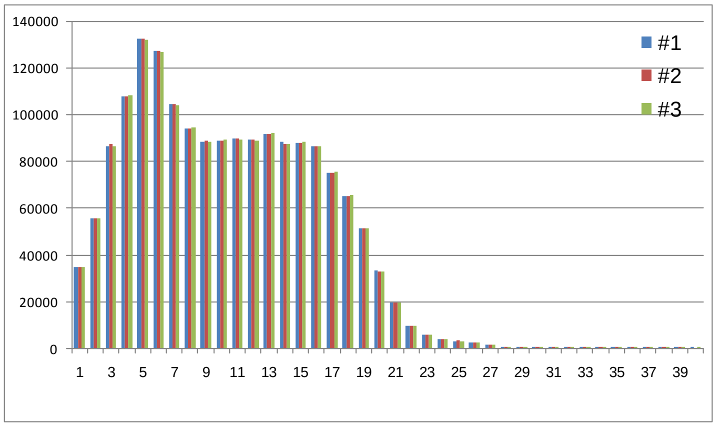


Figure 4: Case-study #1 - comparison of the distribution latency for frame E27 (ECU6 to ECU7) obtained in 3 simulations with different random offsets and different random drifts.

In the following, we will not directly check the convergence of the distributions but this will be done through the value of the Q5 quantiles. Indeed, we are in the context of critical systems mostly interested in the convergence of the tails of the distributions. Q5 is chosen because it remains possible to estimate for a large number of simulations, as required by the experiments, and corresponds to the kinds of constraints one can expect for most automotive functions (see [Na14] for an example).

Whatever the exact initial condition, each of the simulation run led to close estimations of the Q5 values for the different frame. This can be seen on Figure 5 were the Q5 curves obtained in 3 simulation runs are almost superposed for each of the case-study shown. The average difference between the minimum and maximum value of the frame quantiles is below 2.5% for each of the case-study.

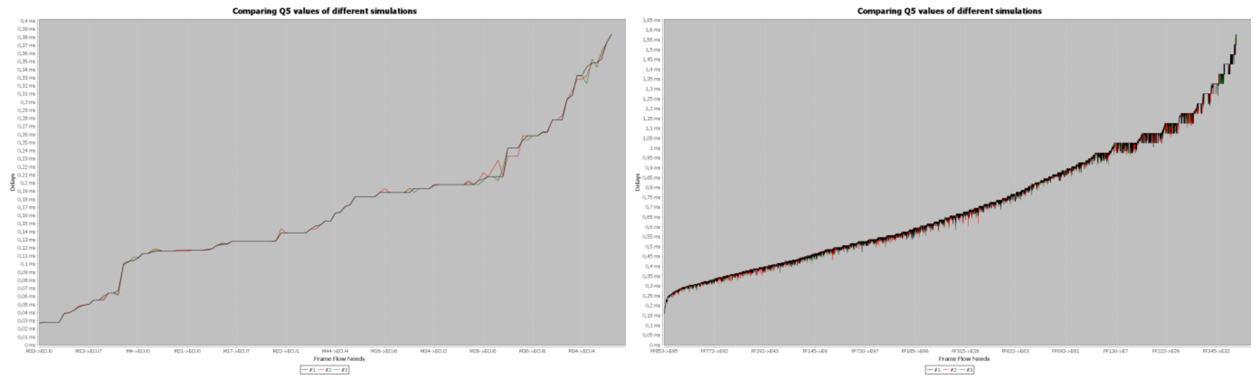


Figure 5: Comparison of the Q5 quantiles of the frame latencies obtained in 3 distinct experiments with different random offsets and different random drifts for case-study #2 and #3. The average difference between the maximum and minimum Q5 value obtained in the 3 experiments ranges from 1.9% to 2.3% in the 3 case-studies.

It should be noted that the points where the curves are not superposed often correspond to frames whose periods are larger than 100ms and thus for which the simulation length may be too short. For instance, in case-study #1 there are several frames with a period equal to 1s, and a large fraction of the frames have a period of 128ms in case-study #2 and #3.

For all three case-studies, we obtain empirical evidence that the simulated network is ergodic. This implies that we do not have to consider the exact initial conditions[3] and that a single long simulation run is sufficient to derive reliable statistics. It also means that it is possible to aggregate the results of different simulations runs done in parallel, if we are interested in the value of higher quantiles such as Q6 or Q7 or if the simulation model is larger. Future work should be devoted to determine what the exact requirements are for a simulation model to remain ergodic. This will enable us to model the embedded systems in a more fine-grained manner by modeling the behavior of higher level protocol layers (e.g. Some IP, see [Se15, Sey15b]), models of ECUs and tasks.

3.3 Q3: what is the appropriate simulation run?

A difficult issue in simulation is to know what the minimum appropriate simulation time is. Indeed, even with a high-speed simulation engine, simulating many variants of large communication architectures, as typically done during the design process, is a time-consuming activity and too much time and computational resources should not be wasted.

This question is discussed in this paragraph in the case where the simulated system is ergodic. A first intuitive answer to this question is that the length of simulation should be long enough so that the start-up conditions do not matter anymore. Indeed, if the simulation time is too short, the transient behavior occurring at the beginning will induce a bias in the statistics (see for instance statistics in the synchronous case in [Na15]). One way to deal with that is to remove the transient period at the beginning from the statistic samples. Although there are heuristics for that, it is not clear-cut to know exactly what defines a transient state and where it ends (see [Cl15] for a recap). The other approach adopted here is to simulate sufficiently long so that the transient state is amortized.

In our experiments with random offsets, samples of quantile values with at least 10 points lead to robust results in the vast majority of cases. In a few cases, statistics out of 20 values were needed and we set this as the requirement in this study. Such sample sizes can be obtained by simulating 2 days of communication for frames with a period lower than or equal to 85ms. The corresponding simulation takes several hours to perform on a single processor, which remains practical for system designers.

[3] The configuration where all nodes start to transmit at the same time, in a synchronous manner, leads to results that are distinctively different from any random startup that we have simulated. The reason underlying this behavior is studied in [Na 15].

4 Scope of application of simulation with respect to schedulability analysis

Simulation is well suited to estimate, early in the design phases, the kind of performances that can be expected in the case of a typical functioning mode of a system. This can be done with a high statistical confidence with the use of higher quantiles of the distributions of the quantities of interest (see [Na14]). Another advantage of simulation, especially when it is coupled with the right analysis and visualization tools, is that it provides a valuable help to understand the behavior of the system in some specific conditions that can be reproduced (see [Na15]). Here we experimentally estimate the extent to which timing-accurate simulation is able to identify the largest possible latencies that can be observed in a switched Ethernet network.

4.1 Q1: are worst-case traversal times computed with Network Calculus accurate?

The pessimism of a schedulability analysis is in general not known and that makes it difficult for the system designer to rely on it for the design choices. In [Ba09], the authors propose a technique to identify for each flow in the system an unfavorable scenario leading to latencies close to the worst-case situation. These unfavorable scenarios provide lower-bound on the WCTTs which can serve to estimate the accuracy of the WCTT upper bound analysis. Indeed, if the real worst-case latency for a flow is unknown, one knows that it lies between the WCTT upper bound and the lower bound. RTaW-Pegase implements an algorithm inspired from the one first proposed in [Ba09]. However, this WCTT lower-bound calculation is not available yet in RTaW-Pegase in the prioritized case, thus it can only be applied on case-study #2.

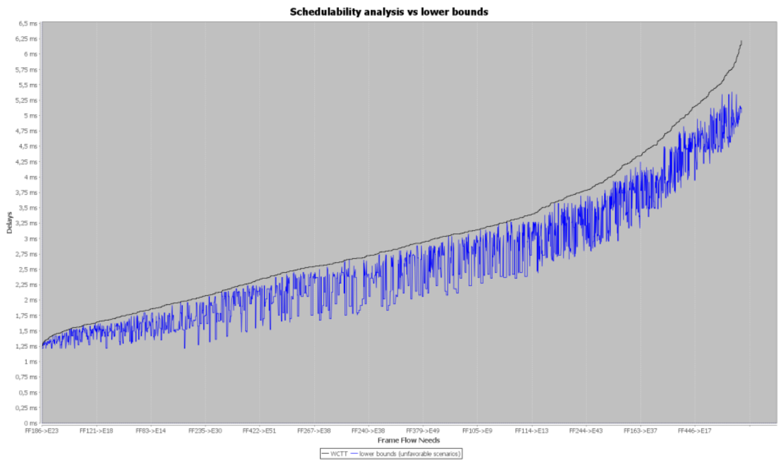


Figure 6: Case-study #2 - upper bounds on the Worst-Case Traversal Times (black curve) shown with a lower bound on the WCTT (blue curve). The flows on the graph are sorted by increasing latencies of the WCTTs, which explains why the lower curve is not monotonous unlike the WCTTs (screenshot from RTaW-Pegase).

The results in Figure 6 show evidence that, except for a small fraction of the flows, the WCTT analysis is accurate: the average difference between the lower bounds and the WCTT upper bounds being on average 4.7% (up to 35% in the worst-case). Similar results were obtained for non-prioritized versions of the two other case-studies, and these results are in line with experiment published in [Bo12] on a different case-study. Though these experiments have been conducted in the non-prioritized case, this result suggests to us that WCTT should also be accurate with priorities.

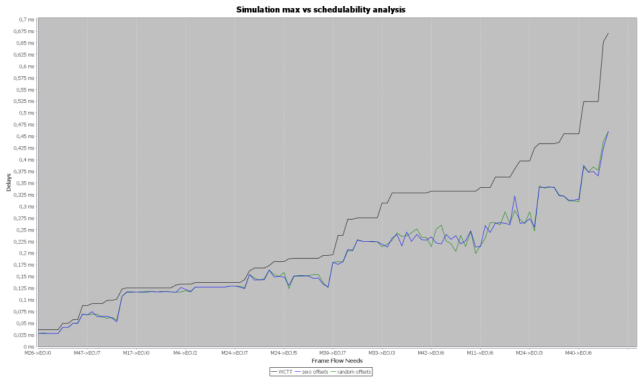
4.2 Q3: maximum observed response times versus worst-case traversal times

The question that is here experimentally investigated is whether the maximum values of the response times that can be observed during a simulation are close to the WCTT upper bounds obtained by analysis. Simulations of the three case-studies have been performed with two clock drift rates (±200ppm and ±400ppm) and various offset configurations:

- 10 random offset assignments where nodes starts within 100ms after the startup of the system,
- a configuration where this startup delay is extended to 1000ms,
- a configuration where all offsets are null, and thus nodes start synchronously (refer to as the zero-offset configuration in the following).

Results discussed here have been obtained with a simulation time set to 8 days of functioning time, The same experiments performed with 2 days of simulation lead to the same results for the two larger case-studies while for case-study #1 simulating 8 days instead of 2 days allowed to decrease the difference with WCTT by 3% both for the average and max value. Whatever the case-study, what we observe is that no random offset assignments lead to significantly different results than the others. For instance, in case-study #1 the average difference for the maximum latency is 6.3% among 10 simulations with distinct random assignments of 2 days. Larger offset intervals and clock drift rates did not make a difference either in our experiments.

In case-study #1, random offsets and synchronous offsets do not behave significantly differently in terms of WCTTs as can be seen in Figure 7 (left graphic), with WCTTs that are on average about 20% less with simulation with respect to schedulability analysis. However a notable difference can be seen on the two larger case-studies when all offsets are set to zero. Explanations for this behavior are discussed in [Na15].



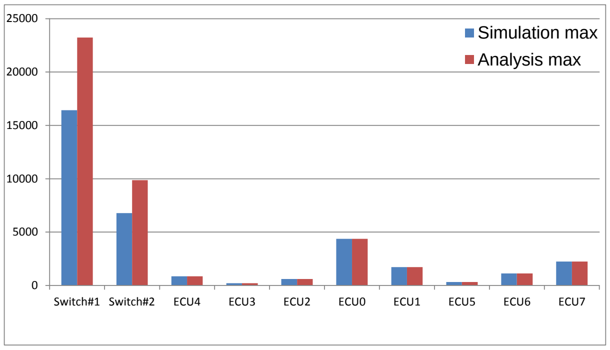
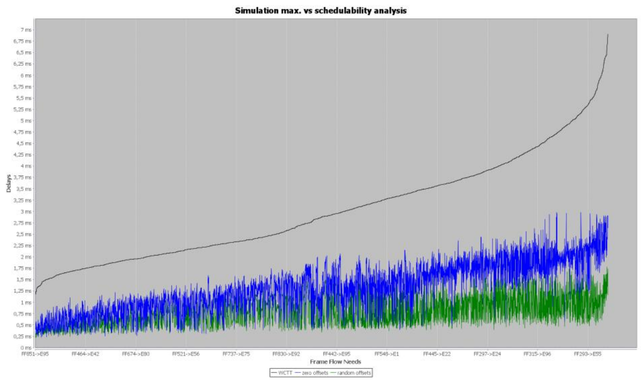


Figure 7: Case-study #1 – Left: upper bounds on WCTTs derived by schedulability analysis (black curve) and the maximum response times observed in simulation. The blue curve is obtained with null offsets for the nodes (i.e., zero-offset configuration), the green curve with random offsets. On average, the simulation with zero offsets leads to response times that are 21% less than the WCTTs, up to 48% maximum (with 5 points above 35%). Right: memory use in switches and nodes (in bytes) as derived by a long simulation (random startup offsets) and schedulability analysis.

Figure 7 (right graphic) shows for case-study #1 the difference between the maximum amount of memory used in the switches/nodes during a long simulation and the upper bounds computed by analysis. Both techniques lead to the same maximum memory usage in the stations, this is because the worst-case situation (one frame for each of the flow of the station) is encountered during each simulation since there are no offsets among the flows of the same station (unlike in [Bo14] for instance where frame offsets are known and accounted for). This observation holds for all case-studies. In case-study #1, the maximum memory usage observed with simulation is at most 31% less (switch #2) than the upper bound calculated by schedulability analysis. This result can be explained by the fact that there is direct relationship between frame latencies and memory size needed in the switches.



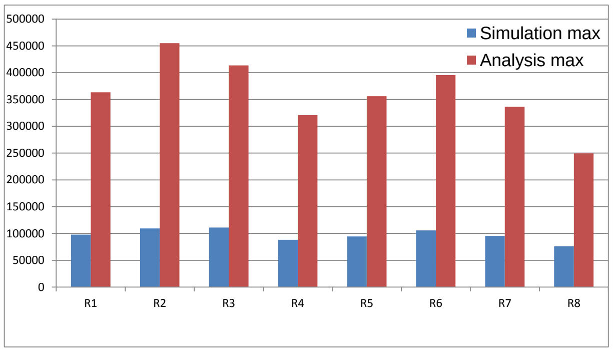


Figure 8: Case-study #3 – Left: difference between upper bounds on WCTTs derived by schedulability analysis (black curve) and maximum response times in simulation (blue curve). The blue curve is obtained with null offsets for the nodes (i.e., zero-offset configuration), the green curve with random offsets. On average, the simulation with zero offsets leads to response times that are 56% less than the WCTTs, up to 88% maximum. Right: memory use in switches (in bytes) as derived by simulation and schedulability analysis.

In case-study #3, the WCTTs obtained by simulation are on average 56% smaller than the WCTTs obtained by schedulability analysis (up to 88% for a given flow). The maximum memory consumption observed with simulation is at most 76% less than the upper bound calculated by schedulability analysis (for switch R2). This result can be explained by the large discrepancy there is between the maximum frame latencies derived by simulation and analysis. However, during this study, we identified sources of pessimism in the memory analysis which can help to reduce the gap between simulation and analytic results.

Our conclusion is that simulation becomes quickly unable to identify large frame response times as the size of the system increases: the *average* difference over all flows ranges from 23% for the smallest case-study to 56% for the largest case-study. The same observation holds equally for the maximum memory consumption in the switches.

Given the length of the simulations and the diversity of experimental conditions in this study, this also suggests that response times close to the WCTT upper bounds are extremely rare events. Indeed, since Q6 values are lower than the WCTT, it means that no more than one frame every million transmissions will experience latencies larger than Q6 and up to the WCTT. In systems, where deadline misses provided that they are rare enough and quantified, can be tolerated, simulation will lead to much more efficient resource usage than schedulability analysis.

5 Conclusions

There are networking technologies such as AFDX or TTP/TTEthernet which have been conceived with the requirement that the temporal behavior of the network must be predictable, if not deterministic, and are thus amenable to worst-case verification with limited pessimism (see [Bo12, Bo14] for AFDX). AUTOSAR-based automotive architectures, based on CAN or Ethernet, are in our experience not as easily analyzable from a timing point of view, because of their complexity, heterogeneous hardware and software components, and because the temporal behaviors of the ECUs and gateways are less constrained.

On the other hand, AUTOSAR offers a wide range of configuration options and complex execution mechanisms to support in an efficient manner the numerous requirements of automotive communications, and the scope of what is possible is still increasing with for instance the introduction of SOME/IP [Vo13] in AUTOSAR. As a result, schedulability analyses for automotive systems are in our opinion not able today to capture the entire complexity of the system with the risk to be pessimistic and possibly unsafe. In addition, it is acceptable for most automotive functions to tolerate occasional deadline misses and message losses, provided that the risk is well quantified and the functions made robust to these events. These two reasons motivate in our view the use of simulation along with schedulability analysis for the design of automotive systems, as this is explored in this paper.

Simulation models when used in the design critical system imposes the use of high-speed simulation engines in order to derive statistical samples of sufficient size for the required confidence level. With the computing power readily available today on desktop workstations, it is possible to simulate complete heterogeneous automotive communication architectures made of CAN, Ethernet and FlexRay buses. However, complete system-level simulations, which would include models of the functional behavior, will require distributing the computation on clusters of machines. Performing simulations on different processors and aggregating the results, would be thus key to allow system-level simulation with high confidence level.

In this work, we have provided empirical evidence that the simulated model of a switched Ethernet network is ergodic and thus that this approach leads to correct results. This question remains however to be answered in a more formal manner and at the scope of a complete electronic embedded architecture. One possibility is to identify the conditions under which the simulation model is equivalent to a Markov Chain and study its ergodicity.

6 References

[AFDX05] "Aircraft data network part 7: Avionics Full DupleX switched Ethernet (AFDX) network", ARINC specification 664P7, June 2005.

[Ba10] H. Bauer, J.-L. Scharbarg, C. Fraboul, "Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach", IEEE Transactions on Industrial informatics, Vol 6, No. 4, November 2010.

[BoTh07] A. Bouillard and E. Thierry, “An algorithmic toolbox for network calculus”, Discrete Event Dynamic Systems, 17(4), october 2007.

[Bo11] M. Boyer, J. Migge, and M. Fumey, “PEGASE – a robust and efficient tool for worst-case network traversal time evaluation on AFDX”, SAE Aerotech 2011, Toulouse, France, 2011.

[Bo11b] M. Boyer, J. Migge, N. Navet, “A simple and efficient class of functions to model arrival curve of packetised flows“, First International Workshop on Worst-case Traversal Time (WCTT), in conjunction with the 32nd IEEE Real-time Systems Symposium (RTSS), Vienna, Austria, November 29, 2011.

[Bo12] M. Boyer, N. Navet, M. Fumey, “Experimental assessment of timing verification techniques for AFDX“, Embedded Real-Time Software and Systems (ERTS 2012), Toulouse, France, February 1-3, 2012.

[Bo14] M. Boyer, L. Santinelli, N. Navet, J. Migge, M. Fumey, “Integrating end-system frame scheduling for more accurate AFDX timing analysis“, Embedded Real-Time Software and Systems (ERTS 2014), Toulouse, France, February 5-7, 2014.

[Bo14b] M. Boyer, L. Fejoz, S. Merz, “Proof-by-Instance for Embedded Network Design: From Prototype to Tool Roadmap“, Embedded Real-Time Software and Systems (ERTS 2014), Toulouse, France, February 5-7, 2014.

[CL15] M. Claypool, “Modeling and Performance Evaluation of Network and Computer Systems - Simulation”, Course CS533, Worcester Polytechnic Institute, 2015.

[Da07] R. Davis, A. Burn, R. Bril, and J. Lukkien, “Controller Area Network (CAN) schedulability analysis: refuted, revisited and revised”, Real-Time Systems, vol. 35, pp. 239–272, 2007.

[Da12] R. Davis, N. Navet, "Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queue", 9th IEEE International Workshop on Factory Communication System (WFCS 2012), Lemgo/Detmold, Germany, May 21-24, 2012.

[It07] J.B. Itier, “A380 Integrated Modular Avionics”, ARTIST2 meeting on Integrated Modular Avionics, 2007.

[Ma13] E. Mabille, M. Boyer, L. Fejoz, and S. Merz, “Certifying Network Calculus in a Proof Assistant”, 5th European Conference for Aeronautics and Space Sciences (EUCASS), Munich, Germany, 2013.

[Ma13b] C. Mauclair, “Une approche statistique des réseaux temps réel embarqués”, Phd thesis from the University of Toulouse, 2013.

[Mo12] A. Monot, N. Navet, B. Bavoux, "Fine-grained Simulation in the Design of Automotive Communication Systems", Embedded Real-Time Software and Systems (ERTS 2012), Toulouse, France, February 1-3, 2012.

[Na14] N. Navet, S. Louvart, J. Villanueva, S. Campoy-Martinez, J. Migge, “Timing verification of automotive communication architectures using quantile estimation“, Embedded Real-Time Software and Systems (ERTS 2014), Toulouse, France, February 5-7, 2014.

[Na15] N. Navet, J. Seyler, J. Migge, "Timing verification of real-time automotive Ethernet networks: what can we expect from simulation?", technical report of the University of Luxembourg, to appear in 2015.

[Na11] N. Navet, A. Monot, B. Bavoux, "Impact of clock drifts on CAN frame response time distributions", 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011), Industry Practice track, Toulouse, September 2011.

[Se13] J. Seyler, "A Tool-Chain for Modeling and Evaluation of Automotive Ethernet Networks", Automotive Bus systems + Ethernet, Stuttgart, Germany, December 9-11, 2013.

[Se15] J. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, “Formal Analysis of the Startup Delay of SOME/IP Service Discovery”, DATE 2015, Grenoble, France, March 9-13, 2015.

[Se15b] J. Seyler, N. Navet, L. Fejoz, “Insights on the Configuration and Performances of SOME/IP Service Discovery“, SAE World Congress, Detroit, USA, April 21-23, 2015.

[SoAd] AUTOSAR, “Specification of Socket Adaptor”, Release 4.2.1, 2015. Available at url http://www.autosar.org/.

[Vo13] L. Völker, “SOME/IP – Die Middleware für Ethernet-basierte Kommunikation”, Hanser automotive networks, 2013.

[Ze09] H. Zeng, M. D. Natale, P. Giusto, and A. L. Sangiovanni-Vincentelli, “Stochastic Analysis of CAN-Based Real-time Automotive Systems,” IEEE Trans. Industrial Informatics, vol. 5, no. 4, pp.388–401, 2009.

[Ze10] H. Zeng, M. D. Natale, P. Giusto, and A. L. Sangiovanni-Vincentelli, “Using statistical methods to compute the probability distribution of message response time in Controller Area Network,” IEEE Trans. Industrial Informatics, vol. 5, no. 4, pp.678–691, 2010.

A Practical Approach to the Simulation of Safety-critical Automotive Control Systems considering Complex Data Flows

Sébastien Dubé
Hella Engineering France
Toulouse, France
Email: sebastien.dube@hella.com

Mesut Özhan
INCHRON GmbH
Potsdam, Germany
Email: oezhan@inchron.com

Achim Rettberg
Hella Electronics
Lippstadt, Germany
Email: achim.rettberg@hella.com

I. Abstract

Embedded systems highly contribute to the efficiency, safety, and usability of our present-day means of transport like cars and airplanes. Due to the possible hazards and risks involved with their operation, safety standards like DO-178C for avionics and ISO 26262 for automotive commend the application of methods and tools according to the state of the art. Functional safety requirements imposed on hardware and software imply the detection of malfunctions and taking corrective actions, before hazards actually occur. As described in [5] one of the key challenges thereby is the prediction and verification of the system's timing behavior. In this paper we describe a model-based approach for real-time simulation focusing on complex end-to-end data flows typically encountered in safety-critical automotive control applications. Based on first-hand experiences gained during the development of an electrical power steering control system, we illustrate how real-time simulation models can be utilized to guide design decisions, and help to achieve safety goals defined at system level. Furthermore, we discuss the issues of response time analysis for dynamic state-dependent data flows considering different semantics for communication in the context of the AUTOSAR standard.

II. keywords

Automotive Control Applications, Electronic Power Steering, Fault Tolerant Time Interval, Response Time Analysis, Real-Time Simulation, AUTOSAR

III. Motivation

Engineers developing electrical / electronic systems for the automotive or avionics domain have to manage a high degree of functional, technological, and organizational complexity. Design and implementation of highly dynamic safety-critical applications make a model-based approach with careful consideration of fault tolerance indispensable. A fault-tolerant design follows a safety strategy which defines the (worst) conditions that a system must cope with, and defines safety mechanisms for fault detection and error handling that must be implemented. Such a design ensures that a system experiencing malfunctions remains operational – possibly with reduced functionality – and transits into a safe error-free state.

Safety mechanisms built into a system are liable to hard real-time requirements. Hence, the transition into a safe state must be achieved under all possible conditions with respect to a limited time span, the so called fault tolerant time interval (FTTI) as defined in [23]. However, if one considers the many different factors that influence a system's timing behavior, e.g. the number of possible faults and failure states, the complexity of the data and control flow, or the scheduling properties of different hardware and software variants, it turns out, that finding a robust and reliable dynamic architecture design is a very challenging task. The problem of verifying the schedulability of processes and messages in a distributed system is NP-hard [7], and besides hard real-time requirements, for safety-critical systems additional requirements must be considered in order to guarantee dependability.

During the past decades, since Liu and Layland in [19] presented their work about *rate-monotonic scheduling* (RMS), research in scheduling theory has developed a large number of concepts and mathematical proofs in order to take on the challenges presented by the analysis and optimization of embedded control applications. But, despite the many advances in extending their scope and applicability as described by Sha et al. in [22], most theoretical approaches remain limited to specific use cases as typically some of their constraints are violated in real industrial systems. Examples for those limitations and constraints are given by Cooling in [9] and Davis et al. in [10]. A more recent study taking into account precedence relations between tasks is provided by Kermia in [16].

The software standards developed by the Automotive Open System Architecture (AUTOSAR) partnership [3] have dramatically changed the way automotive systems are built today, and the transformation of processes, methods, and tools is not yet finished. Since AUTOSAR release 4.0 there is also a dedicated specification [4] for the formalized description of timing properties and constraints available. And although, this helps to identify and exchange timing-related information in a complex automotive supply chain, AUTOSAR does not address the issue of how this information can be obtained, or a timing analysis can be performed.

In order to evaluate a system's performance, interoperability, robustness, and eventually its safety, the specific characteristics of the AUTOSAR operating system and run-time environment (RTE) must be considered on implementation level. These comprise concepts like OS applications, IOC, shared (multi-core) resources, schedule tables, and extended tasks. For AUTOSAR compliant real-time systems, Anssi et al. in [1] presented a study, where the applicability of schedulability analysis is evaluated using an open source implementation [12] of Palencia's and Harbour's algorithm [21]. Additional evaluations are provided by Hladik et al. in [15].

Viewed from the system engineering perspective eventually one key issue remains: Even if state of the art scheduling analysis methods can be applied, they can only provide true or false statements on the feasibility of a given system configuration. However, in order to perform design modifications efficiently, engineers need more fine-grained information about their system's dynamic behavior, e.g. obtained by statistical analysis of trace data coming from target measurements or real-time simulation.

IV. Case study: Electronic Power Steering

The research results presented in the following are based on the observations and experiences made by the authors during the development of an electronic power steering (EPS) system at Hella Engineering in Toulouse, France. Due to the safety and hard real-time requirements for this system, a model-based approach using SysML at logical architecture level, and AUTOSAR methodology at technical architecture level was implemented. For the simulation, visualization, and exploration of various design alternatives methods and tools [14] provided by INCHRON were also used from the very beginning of this project.

A. System overview

The electronic power steering system in our case study as depicted in figure 1 uses a brushless motor to assist the driver in steering his vehicle. Position and torque of the steering column are permanently measured by sensors and processed by the *steering control module* (SCM), which calculates an assistive torque that is applied depending on different driving conditions. Advantages of an electronic power steering system over a comparable hydraulic solution are improved fuel efficiency and greatly simplified manufacturing and maintenance processes. Also, in combination with an electronic stability control it vastly contributes to improved driving safety.

B. Objectives

A major challenge during the development of embedded systems such as the electronic power steering is the verification of end-to-end latency requirements. The difficulty lays in the fact that these systems feature many different functional and dysfunctional modes of operation with corresponding hard real-time requirements for monitoring, error detection, and error handling. Depending on the required safety level, the implementation of these safety mechanisms in addition to the actual control functionality, drastically increases the complexity of the data and control flow. As a consequence for the EPS system development, following a classical integration and test approach solely based on measurements on the target hardware, seemed not feasible. In order to reduce the time and effort, that it takes to find resource bottlenecks, timing

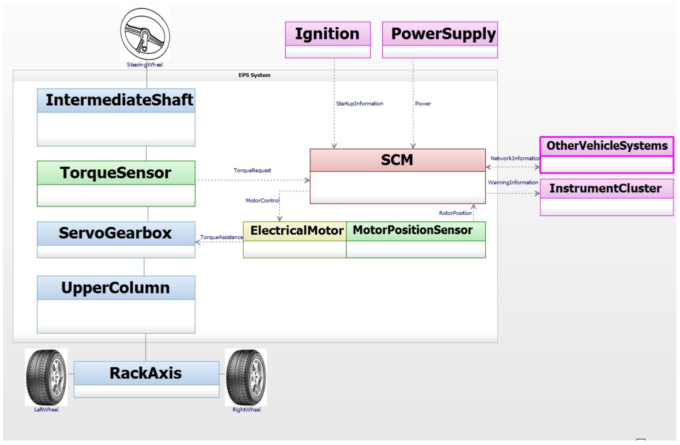


Figure 1. Electro-mechanical components of the electronic power steering system

errors, and eventually to verify the real-time requirements, it was one of our main objectives in this case study to apply state-of-the-art methods and tools, aiming at a virtual integration of the system in earlier development phases. Furthermore, the verification of an embedded system requires that timing properties and constraints are specified in a formal, unambiguous way, matching with the semantics of the formalism which is used for the analysis. Thus, another objective was to check if all the system information, needed by either formal schedulability analysis or model-based scheduling simulation, is available in a real project environment. Finally, we wanted to achieve a seamless integration of all methods and tools avoiding redundant modeling of the same information as much as possible.

C. Design of the dynamic architecture

An important part of the dynamic architecture design for the SCM was planning the execution of tasks and interrupt service routines (ISR) on the main microcontroller. The initial design was created according to the strategy of *deadline-monotonic scheduling* (DMS) [18] where tasks are assigned priorities depending on their deadline with priorities being inversely proportional to the length of the deadline. Compared to RMS, deadline-monotonic priority assignment is an optimal strategy for tasks that can have a deadline equal or smaller than their activation period. Furthermore, Audsley and Burns [2] [7] showed that with additional schedulability tests, the deadlines of sporadic tasks can be guaranteed within the deadline-monotonic theory.

Table I shows the initial scheduling configuration for the SCM according to the deadline-monotonic priority assignment strategy.[1] In order to preserve most of the scheduling characteristics guaranteed by DMS, the SCM design was restricted to only use time-triggered, basic tasks with no priorities shared between two different tasks.

For the model-based simulation with the INCHRON Tool-Suite a project file (.ipr) of this configuration needed to be

[1]Note, that due to intellectual property rights of the companies involved in the development of the SCM, only anonymized and simplified versions of the original system information can be documented.

Table I. Scheduling with OS counter tick of 1ms

Name	Type	Period (μs)	Deadline (μs)	Offset (μs)	Priority
Torque_Manager	Task	1000	100	async	200
Analog_Manager	Task	1000	500	0	102
ASIC_Manager	Task	1000	1000	0	95
Mode_Manager	Task	5000	5000	2000	70
Com_Rx	Task	5000	5000	2000	50
Com_Tx	Task	5000	5000	2000	45
Temp_Manager	Task	10000	10000	4000	20
Memory_Manager	Task	10000	10000	9000	5
Diagnosis	Task	10000	10000	8000	2

created. As suggested by the TIMMO-2-USE project in [11], there are basically two different possibilities to create a model for the simulation: either top-down based on requirements and design specifications in an early project phase, or bottom-up by reverse-engineering the necessary information based on an existing (prototype) implementation of the system. In this case study the initial simulation model was generated by importing the AUTOSAR ECU configuration data of a prototype sample into the INCHRON Tool-Suite.

Another very important input parameter required for the schedulability analysis and simulation is the core execution time (CET) of the scheduled processes. Obtaining actual and reliable execution time information in reality is not an easy task. Even if sophisticated measurement capabilities are available, it can be very difficult or nearly impossible to do measurements for all possible operation states and (error) conditions in many different variants. An alternative to measurements on the target hardware is to predict worst case execution times by using static code analysis, e.g. as suggested by Ferdinand in [13]. However, this approach also has its limitations: for example increasingly complex caching and pipelining solutions found in modern microcontrollers, make it very difficult to predict the time consumption of machine code instructions.

In the case study, the model-based simulation of different scheduling scenarios was performed by using execution time information which was measured on the target hardware. Measurements were taken for various operation states, whereas focus was laid on those configurations that covered the most complex execution paths (in case of the control functions). In order to further increase the confidence into the results, we applied scaling factors to some of the measured values for those functions where behavioral details were unknown. However, it should also be noted, that for the verification and optimization of end-to-end latencies in a distributed system, time delays caused by asynchronous process executions can be more important, than the deviation of individual execution times.

D. Event chains with hard real-time latency requirements

A temporally ordered sequence of correlated events, that can be observed or measured in a system, is referred to as a chain of events, or event chain. Applied to embedded real-time systems, the concept of an event chain can be used to specify a sequence of function executions and (communication) data flows between them, which are subject to safety and real-time requirements. Figure 2 shows an example for such an event chain in the SCM, starting with the sampling of the torque sensor and ending with the control of the motor realizing the steering assistance.

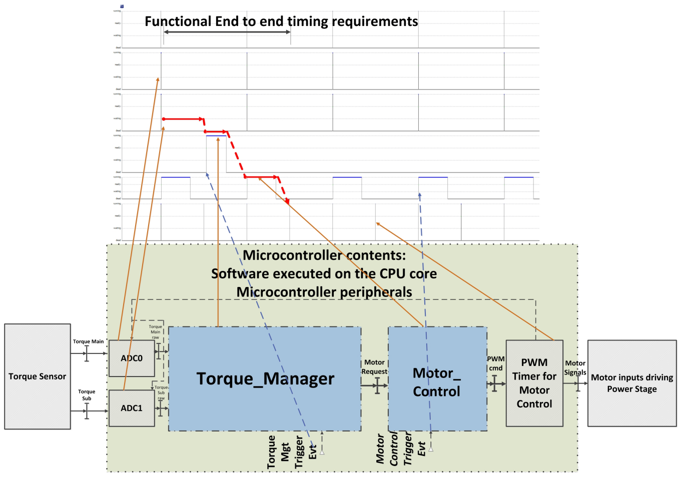

Functional End to end timing requirements
Microcontroller contents:
Software executed on the CPU core
Microcontroller peripherals
Torque Sensor
Torque Main
Torque Sub
ADC0
ADC1
Torque Main raw
Torque Sub raw
Torque_Manager
Motor Request
Motor_ Control
PWM cmd
PWM Timer for Motor Control
Motor Signals
Motor inputs driving Power Stage
Torque Mgt Trigger Evt
Motor Control Trigger Evt

Figure 2. Event chain with end-to-end latency requirement

The event chain concept is a very useful abstraction in order to describe the scope of an end-to-end latency requirement from the perspective of a control or system engineer, considering the influence of both the hardware and the software. In the automotive industry it is a widely used concept, but only since its formalization by the AUTOSAR standard, the semantical pitfalls of ambiguous textual and visual descriptions could be mitigated.

A key issue in the formal semantics of an event chain concerns the definition of data flow properties: the data flow in a system results from the production, transmission, and consumption of data by different hardware and software functions. In a distributed system the deployment (location) of a function mostly determines the means by which this function is able to communicate with other local or remote functions, e.g. via messaging over a network, shared variables etc. As the interpretation of an event chain depends on the behavior of the functions in its scope, and again their behavior depends on the value, state, order, and age of the processed data, a formal semantics of the data flow needs to take the different communication means and their characteristics into account. At the same time, the mathematical model which is used by the analysis or simulation, must be able to handle that formalism.

Initially, at the beginning of the case study, the data flow between hardware and software components at system level was modeled by using the *flow port* concept of SysML [20] as depicted in the figure 3. This modeling view helps to understand the relationships between architectural blocks, differentiated between hardware and software. However, some essential data flow characteristics as described above, were not supported by that concept. Furthermore, we also tried to derive a communication model from the AUTOSAR meta-model, but that approach turned out to be too complex, as the specification of the communication and the dependencies between basic software modules is significantly different from the concepts used for the application software. A further restriction was the difficulty to adequately describe the behavior and influence of hardware functions.

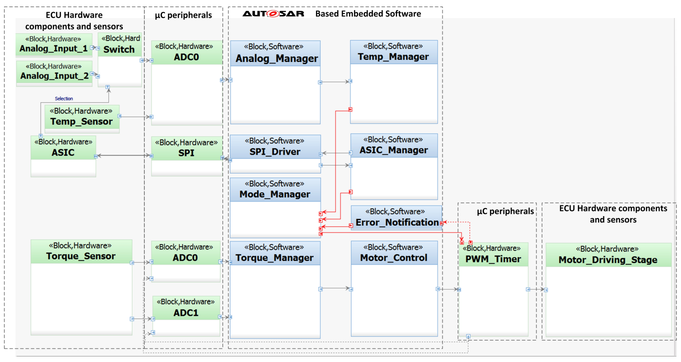


Figure 3. Simplified SCM model as SysML internal block diagram

Eventually, in the case study a rather simple but sufficiently powerful formalism, derived from the co-design methodology for embedded systems (*MCSE*) [8], was used for the specification of data flows as follows:

- **Shared variable** (or permanent data): Data which can be read / written at any time. In order to avoid data inconsistencies, the access to shared variables must be protected against simultaneous read and write operations by asynchronous processes.
- **Event**: An (activation) event that triggers the execution of a process.
- **Queued**: Data is buffered in a queue with FIFO mechanism where the oldest data in the queue is read first. In control theory the queue size usually directly depends on the execution periods (frequency) of the producing and consuming processes.

Figure 4 shows the application of the co-design methodology for the modeling of the SCM system architecture.

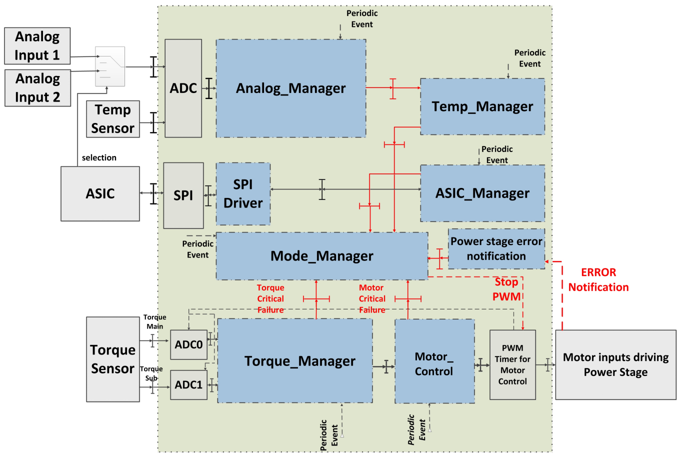


Figure 4. Technical architecture of SCM with functional and dysfunctional data flows

This model describes the main hardware and software functions, and also the data and control flow connections according to the MCSE semantics. Compared to the basic SysML semantics (see figure 3), the most important differences are as follows:

- **Rich data flow semantics**: MCSE supports different types (shared variable, event, queued) for data flows, which allows abstract modeling of the most common communication mechanisms encountered in real systems.
- **Activation flow**: Event-triggered activation of processes is explicitly supported by MCSE, but not the basic SysML model.
- **Timing properties**: Task periods, interrupt interarrival times, and other timing parameters can be specified as properties of a (software) block.
- **Closer to AUTOSAR model**: MCSE is closer to the AUTOSAR meta-model (the SCM is based on AUTOSAR), and offers better support for the concepts used in the specification of the software architecture.

In the case study, this model was used as a starting point for the definition of the data and control flow in the INCHRON Tool-Suite. Basically, the MCSE concepts could be mapped one-to-one to semantically equivalent communication concepts in the tool. A further advantage of this approach is that event chain definitions for the functional and dysfunctional operation states of the system can be described in a common model, using the same data and control flow concepts. For this paper, we have selected two (dysfunctional) event chains, in order to demonstrate the application of our approach for the verification of end-to-end FTTI requirements. Figure 5 shows the sequence of process executions for each event chain similar to the definition in the simulation tool.

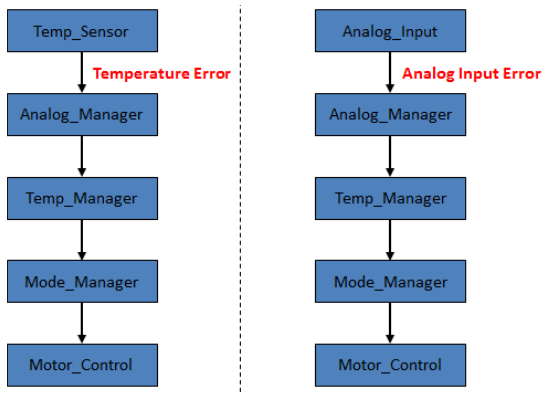
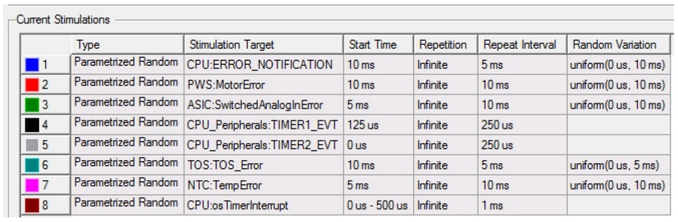


Figure 5. Event chain definition for *Temperature Error* and *Analog Input Error*

They start with the detection of a failure at different signal sources, but then follow the same sequence of function executions until they end with the transition into a safe state (here disabling the application of the motor torque to the motor power stage). Arrows between the processes represent a (typed) data flow connection in the model, which are then traced and highlighted in the simulation (see figures 10, 11, 13, 14).

In general, a failure of the system can have different underlying causes originating from either the environment (e.g. the system operating under conditions different from its specification) or the system itself (e.g. a malfunction of an electrical, electronic, or mechanical component). However, at software level for the verification of FTTI requirements, this difference can be neglected. For example, if the system needs to shutdown when sensors indicate a too high temperature, we do not need to care, if this indication is caused by a sensor malfunction, or the system temperature being actually too high. Regardless of the failure source, the dynamic architecture design must ensure, that monitoring, error detection and error handling processes are executed in a deterministic way, and meet the FTTI requirements.

In a real system the malfunction causing the system failure can occur at any time, and in order to verify the compliance of the dynamic software architecture with the FTTI requirements, it is necessary to recreate the different scheduling situations in the simulation. Therefore, it is not sufficient to induce an error just once, but it must be done repeatedly over the time of the simulation. For the SCM the stimulation generator of chronSIM was used to define an environment model (scenario), that would randomly induce an error with respect to the activation period of the concerned monitoring or error detection process. For example, if the error detection is executed every $10ms$, then a uniform random variation between 0 and $10ms$ was chosen for the error induction. Figure 6 shows the stimulation scenario defined for the different errors induced during the simulation.


Current Stimulations

	Type	Stimulation Target	Start Time	Repetition	Repeat Interval	Random Variation
🟦 1	Parametrized Random	CPU:ERROR_NOTIFICATION	10 ms	Infinite	5 ms	uniform(0 us, 10 ms)
🟥 2	Parametrized Random	PWS:MotorError	10 ms	Infinite	10 ms	uniform(0 us, 10 ms)
🟩 3	Parametrized Random	ASIC:SwitchedAnalogInError	5 ms	Infinite	10 ms	uniform(0 us, 10 ms)
⬛ 4	Parametrized Random	CPU_Peripherals:TIMER1_EVT	125 us	Infinite	250 us	
⬜ 5	Parametrized Random	CPU_Peripherals:TIMER2_EVT	0 us	Infinite	250 us	
🟦 6	Parametrized Random	TOS:TOS_Error	10 ms	Infinite	5 ms	uniform(0 us, 5 ms)
🟪 7	Parametrized Random	NTC:TempError	5 ms	Infinite	10 ms	uniform(0 us, 10 ms)
🟫 8	Parametrized Random	CPU:osTimerInterrupt	0 us - 500 us	Infinite	1 ms	

Figure 6. Stimulation scenario defined with the INCHRON Tool-Suite

The main benefit of this approach is, that it allows to reach a high coverage of the various, relevant preemption situations in a very short time – much shorter than in a hardware-in-the-loop (*HIL*) or prototype test environment.

V. Simulation and Optimization

Using the INCHRON Tool-Suite we performed several simulation runs in order to compare alternative scheduling configurations for the SCM. The traces generated by the simulation were evaluated according to the following quality criteria:

- Deadline violations
- Response time distribution
- CPU peak load in certain averaging intervals
- Start-to-start jitter
- End-to-end latencies of dedicated event chains

According timing requirements were specified directly in the simulation tool, and evaluation statistics were automatically generated after each iteration. After analyzing these results, the (scheduling) configuration was modified manually, and a new iteration was started. The most important observations and conclusions made during the case study, are presented in the following.

A. Adjustment of start offsets

Figure 7 shows an excerpt from the simulation trace of the initial system configuration as specified further above in table I: process executions (indicated by green areas within the rectangular process box) and preemptions (white areas) by higher priority processes are depicted as a Gantt diagram.

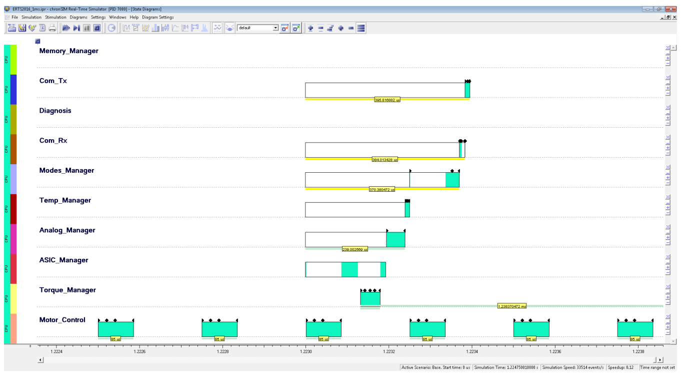


Figure 7. Gantt chart view of processes inside the SCM

One observation made during the analysis of simulation results for the initial scheduling configuration was, that CPU load peaks can occur when processes with relatively large execution times are activated at the same time. For example the time-triggered processes *ASIC_Manager* and *Analog_Manager* both with a period of $1ms$ were affected. Furthermore, *Com_Tx* and *Com_Rx* with a period of $5ms$ also had this issue. An obvious solution to smooth the peak load, would be to introduce a time delay (offset) between the activations of the processes in question. For example for the *ASIC_Manager* task, an activation offset of 500μs against the *Analog_Manager* task seemed feasible.[2] However, the granularity of the underlying OS counter tick was just $1ms$, and thus shifts between the activations were only possible in steps of $1ms$. For this reason, the OS configuration was modified, and the granularity of the OS counter tick was reduced to 500μs.

Table II. Adjusted start offsets for *ASIC_Manager* and *Com_Tx*

Name	Type	Period (μs)	Deadline (μs)	Offset (μs)	Priority
…					
ASIC_Manager	Task	1000	1000	500	95
…					
Com_Tx	Task	5000	5000	3500	45
…					

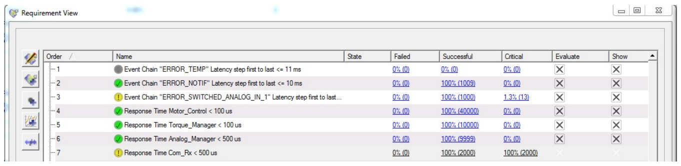
The activation offsets for *ASIC_Manager* and *Com_Tx* were adjusted as shown in table II.

B. Deadline violations

In the case study deadline requirements were defined for all processes. These were monitored and checked during the

[2]The worst-case response time determined for the ASIC_Manager task was lower than 500μs.

simulation of the different scheduling configurations. The simulation tool shows the evaluation of all (response time and other) requirements in a dedicated requirements evaluation view, where the number of successful, critical, and failed checks for each requirement is summarized (see figure 8). A concrete response time check is considered *critical*, if a certain predefined margin relative to the actual deadline is exceeded.



Requirement View

Order	Name	State	Failed	Successful	Critical	Evaluate	Show
1	● Event Chain "ERROR_TEMP" Latency step first to last <= 11 ms		0% (0)	0% (0)	0% (0)	☒	☒
2	✓ Event Chain "ERROR_NOTIF" Latency step first to last <= 10 ms		0% (0)	100% (1009)	0% (0)	☒	☒
3	! Event Chain "ERROR_SWITCHED_ANALOG_IN_1" Latency step first to last...		0% (0)	100% (1000)	1.3% (13)	☒	☒
4	✓ Response Time Motor_Control < 100 us		0% (0)	100% (40000)	0% (0)	☒	☒
5	✓ Response Time Torque_Manager < 100 us		0% (0)	100% (10000)	0% (0)	☒	☒
6	✓ Response Time Analog_Manager < 500 us		0% (0)	100% (9999)	0% (0)	☒	☒
7	! Response Time Com_Rx < 500 us		0% (0)	100% (2000)	100% (2000)		

Figure 8. Evaluation of response time requirements

For example, we can see that the response time of *Com_Rx* is considered critical, as it exceeds the safety margin. Overall, we observed no hard deadline requirement violations in the case study.

C. Execution jitter

In order to monitor the efficiency of a specific scheduling configuration, it may be necessary to monitor additional process statistics. A very useful indicator is the execution jitter. Figure 9 for example shows the time distribution of the start-to-start (purple bars) and terminate-to-terminate jitter (green bars) for the *Com_Rx* task.

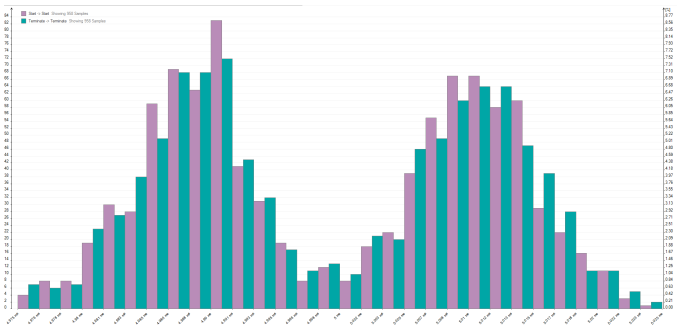


Figure 9. Distribution of execution jitter (start and terminate) for task *Com_Rx*

In the histogram, we can see that the start time of *Com_Rx* fluctuates between 4.98 and $5.03ms$ due to preemptions caused by interrupts and higher priority tasks.

D. Evaluation of event chain 'Temperature Error'

A *Temperature Error* is indicated by the task *Temp_Manager* when the temperature value measured by the temperature sensor exceeds a predefined threshold. If this is the case, the *Mode_Manager* task shall switch into a corresponding failure mode, and send an error notification *PWM_Stop* to the power stage which is driving the motor. According to the system specification, the end-to-end latency requirement goal for this event chain was $12ms$.

The simulation of the SCM with chronSIM predicted, that violations of the end-to-end latency requirement are possible, although no activation violations occur and all processes meet their deadline requirements. An example from the simulation which shows such a requirement violation is given in figure 10.

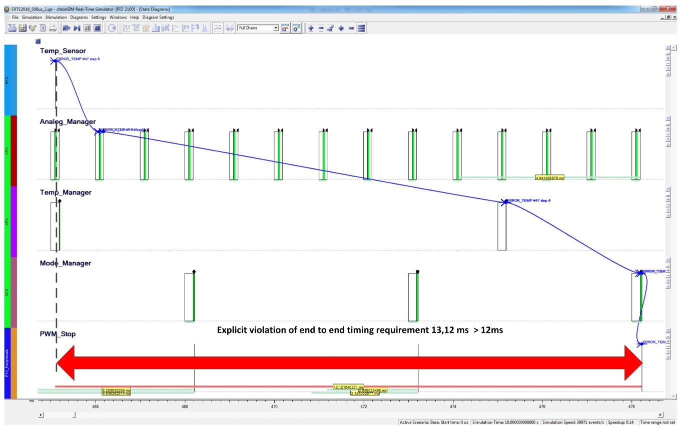

Temp_Sensor
Analog_Manager
Temp_Manager
Mode_Manager
PWM_Stop
Explicit violation of end to end timing requirement 13,12 ms > 12ms

Figure 10. Violation of end-to-end latency requirement (*Temperature Error*) detected in the simulation

In this particular case, we can observe that the end-to-end latency mainly arises from the $10ms$ period of the *Temp_Manager* task, and from the additional delay (almost $3ms$) between the execution of the *Temp_Manager* and the *Mode_Manager* task. One possible solution would be to increase the execution frequency of the *Temp_Manager* task, in order to reduce the time delay which occurs after the provision of new sensor values by the *Analog_Manager* task. However, this would also increase the CPU load.

Alternatively, a solution would be to reduce the time delay between the activation of the *Temp_Manager* and *Mode_Manager* task, by adjusting the start offsets. Furthermore, it is also necessary, that *Temp_Manager* gets a higher priority than *Mode_Manager*, so it is scheduled first. Although, this priority change contradicts the DMS strategy – *Temp_Manager* has a bigger deadline than *Mode_Manager* – it seems to be the most feasible solution to reduce the event chain latency.

Table III. ADJUSTED SCHEDULING WITH OS COUNTER TICK OF 500US

Name	Type	Period (μs)	Deadline (μs)	Offset (μs)	Priority
Torque_Manager	Task	1000	100	-	200
Analog_Manager	Task	1000	500	0	102
ASIC_Manager	Task	1000	1000	500	95
Temp_Manager	Task	10000	10000	2000	71
Mode_Manager	Task	5000	5000	2000	70
Com_Rx	Task	5000	5000	2000	50
Com_Tx	Task	5000	5000	3500	45
Memory_Manager	Task	10000	10000	9000	5
Diagnosis	Task	10000	10000	8000	2

After the adjustment of the scheduling as defined in table III, a rerun of the simulation shows, that the end-to-end latency of the event chain now remains under the deadline of $12ms$. An example for a successful evaluation of the requirement is shown in figure 11.

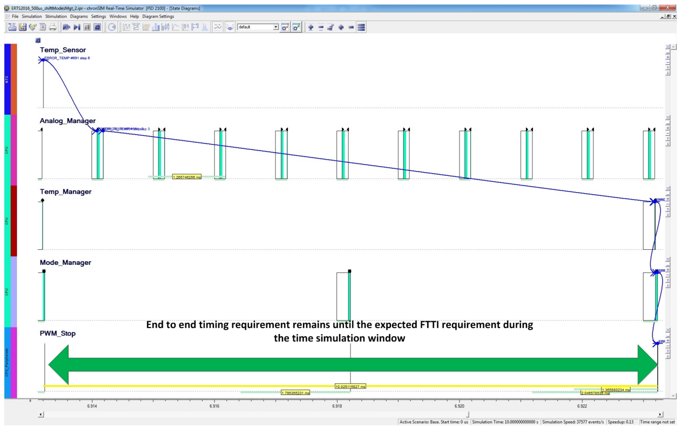

Temp_Sensor
Analog_Manager
Temp_Manager
Mode_Manager
PWM_Stop
End to end timing requirement remains until the expected FTTI requirement during the time simulation window

Figure 11. Successful evaluation of end-to-end latency requirement (*Temperature Error*)

The distribution of the end-to-end latency for the event chain *Temperature Error* predicted by the simulation is shown in figure 12.

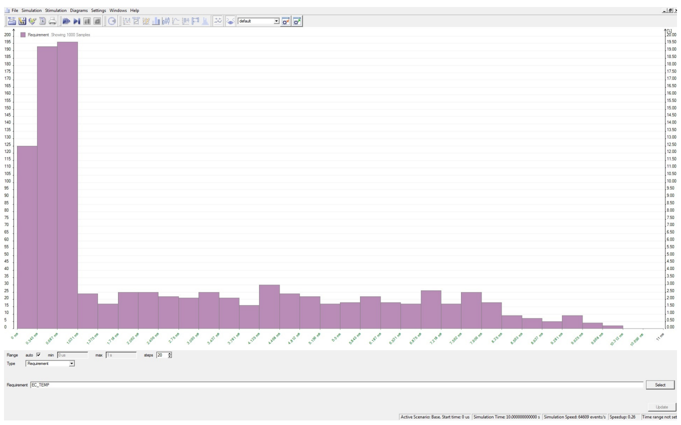


Figure 12. End-to-end latency distribution for event chain (*Temperature Error*)

According to these results, the latency of the event chain ranges from $0.3ms$ up to $10.4ms$.

E. Evaluation of event chain 'Analog Input Error'

In addition to the main temperature sensor, the SCM processes temperature information from two additional temperature sensors in different locations. These are connected to the SCM via two multiplexed (switched) analog inputs. The multiplexing is controlled by an *ASIC* using synchronous SPI bus communication. Depending on the SPI pin selection controlled by the *ASIC_Manager*, the *ADC* reads the sensor signal from either *Analog_Input 1* or *Analog_Input 2*.

As shown in figure 13, the switching between the analog inputs every $6ms$ can lead to a delay for the recognition of a potential *Analog Input Error*. In this situation, we can see that an input error propagating from *Analog_Input 1* is not processed immediately by the *Analog_Manager* task, but only after an additional switching cycle, when *Analog_Input 1* is selected again after *Analog_Input 2* was read. Considering the simulation results we can see that the switching frequency is highly relevant for the optimization of the end-to-end event

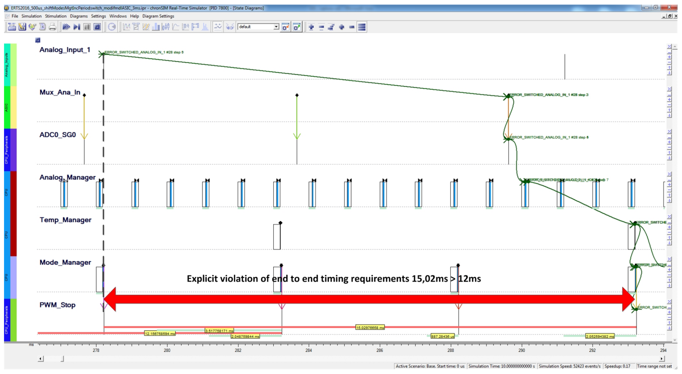


Figure 13. Violation of end-to-end latency requirement (*Analog Input Error*)

chain latency. If we increase the frequency (decrease the period) switching inputs every $1ms$, thus accepting a slight increase of the CPU load, we can reduce the time delay for the error indication to propagate and reach the power stage in less than $12ms$ as depicted in figure 14.

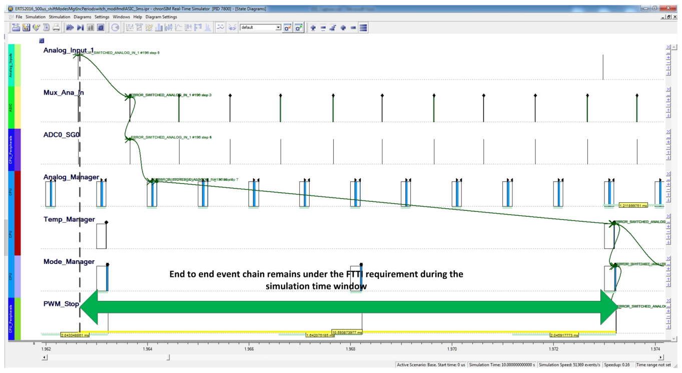

Analog_Input_1
Mux_Ana_In
ADC0_SG0
Analog_Manager
Temp_Manager
Mode_Manager
PWM_Stop
End to end event chain remains under the FTTI requirement during the simulation time window

Figure 14. Successful evaluation of end-to-end latency requirement (*Switched Analog Input Error*)

VI. Comparison of CPU load for scheduling alternatives

Finally, after the optimization of the end-to-end latencies for the different event chains, we compared the CPU load characteristics of the different scheduling alternatives, as described in table I and III.

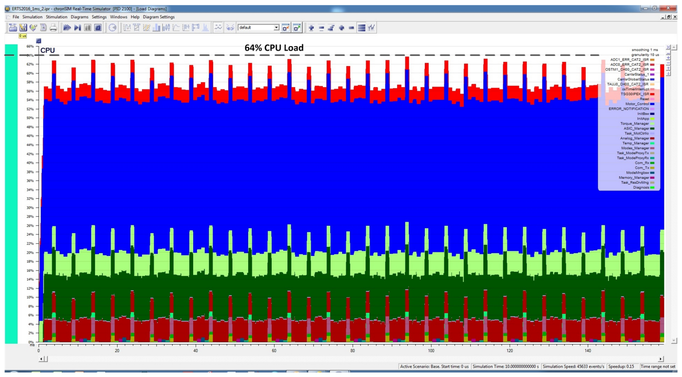


Figure 15. Simulated CPU load average for initial configuration

Figure 15 shows the CPU load for the initial configuration of the SCM before the optimization, and figure 16 shows the CPU load for the final configuration. In both cases the CPU load curve was calculated for a smoothing interval of $1ms$ with a granularity of $10us$.

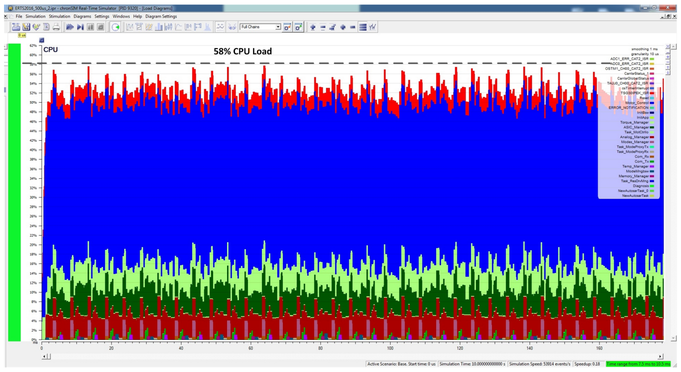


Figure 16. Simulated CPU load average for final configuration after optimization

We can observe that the CPU load peaks in the initial configuration reach up to 64%, and are slightly higher than those observed in the optimized configuration. The decrease from 64% to 58% was achieved mainly by the re-adjustment of the start offset for the *ASIC_Manager* task.

VII. Generic design rules learned from the case study

A key question for the optimization of the scheduling in the SCM was to know which configuration parameters can be changed, and which changes are the most effective in order to fulfill the timing requirements. The theory for preemptive fixed-priority scheduling usually focuses on the optimization of process priorities and relative activation offsets. As we have shown in this paper, the optimization of offsets – even when it is done manually – is very effective to minimize the number of context switches[3] and to smooth the CPU load in time intervals which show high CPU load peaks. At this, it is important to follow the precedence relations imposed by the data flow in the system, as otherwise the end-to-end latencies of event chains will be unnecessarily prolonged. However, we have also shown that for the optimization of event chains, it can be very useful to consider changing also other aspects of the system configuration, like activation periods, or the number and decomposition of processes:

- *Periods of time-triggered processes*: Usually, the activation period of a time-triggered processes is deduced from the system requirements of the functions allocated to this process. Nevertheless, in some cases the activation period can be relaxed without violating any constraints. For example in many control systems, processes concerned with the sampling and (pre-)processing of sensor data are usually scheduled with a higher frequency than necessary, in order to compensate for unpredicted scheduling effects.
- *Number and decomposition of processes*: The number of processes and their decomposition, as a matter of fact the allocation of basic and application software functions (or runnables) to processes, is guided by both functional and safety requirements. As the end-to-end latency of an event chain results from a predefined, sequential chain of function executions and data flows, and thus depends on the timely interaction of the involved processes, it can only be optimized under consideration of the underlying process architecture. In a complex control system, consisting of many event chains with different criticality, designers must find a trade-off between the separation of concerns driven by safety requirements, and the compliance with real-time requirements imposed by the functional domain.
- *Execution order*: In general, the execution order of functions within one process should follow the data flow between the functions. In some cases, additional design techniques must be employed to break up feedback loops in the data flow.
- *Core affinity*: On multi-core processors, the core affinity of a process defines on which core(s) this process is allowed to execute. In order to avoid expensive cross-core communication between processes, functions of the same event chain should not be allocated to processes which execute on different cores.

[3]A context switch in an AUTOSAR OS with memory protection may consume several microseconds.

VIII. Extended timing-aware co-design methodology

Based on the experiences made in the case study, we have enhanced our development process for the verification of (safety-critical) event chains with hard real-time requirements. A seamless workflow as depicted in figure 17 combining timing measurements on the target hardware with model-based timing simulation was defined, and feasibility of the approach was tested using the commercial tools chronVIEW and chronSIM developed by INCHRON.

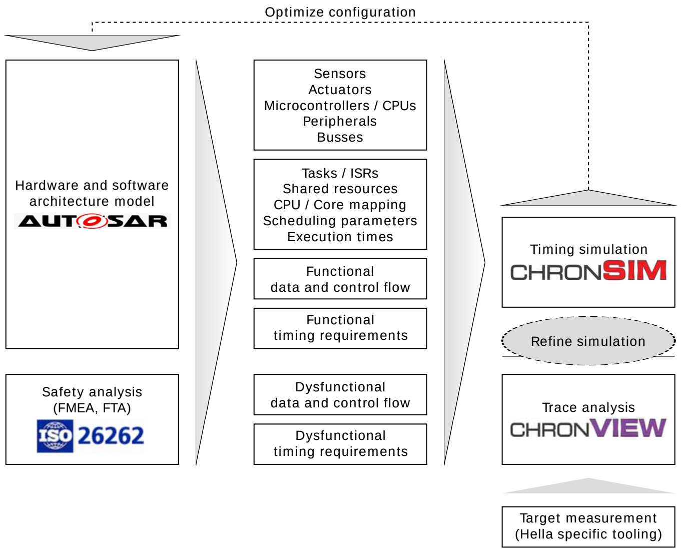


Figure 17. Model-based timing simulation of AUTOSAR compliant systems

The following work tasks shall be performed iteratively:

- Specify the structural system architecture comprising the basic hardware and software elements using the MCSE modeling concepts.
- Identify the interactions between application and basic software (e.g. required services), and basic software and hardware peripherals (e.g. sensor data acquisition).
- Specify the dynamic system architecture comprising the basic execution and communication blocks (tasks, interrupts, messages) and their associated timing properties (BCET, WCET, period, offset, deadline, priority etc.).
- Specify the functional data and control flow considering all relevant modes of operation.
- Perform safety analysis and deduce the dysfunctional data and control flow for all relevant fault conditions.
- Create (using chronSIM's model editor directly) or generate (using the INCHRON python API) a timing simulation for the chronSIM tool out of the various architecture models.
- Define simulation scenarios in chronSIM for the various modes and fault conditions and perform simulation runs.
- If available perform measurements of the integrated target and import and analyze the measured trace with the chronVIEW tool.
- Extract timing properties and update simulation parameters with measured values. This is already done automatically by the tool.
- Iteratively perform simulations and target measurements until all functional and safety requirements can be fulfilled by the current set of timing properties.

IX. Conclusion and future work

Following the practical approach described in this paper, we have shown how state-of-the-art model-based simulation techniques can be used to support the dynamic architecture design of complex automotive control systems. Although, it may appear that adjusting the scheduling configuration in the presented examples is not too complicated, and the proposed solutions may seem obvious, one should consider the number and complexity of the entire event chains in the SCM. In reality, system engineers and software architects responsible for integration and testing, have to achieve many different competing, and sometimes contradicting design goals, especially concerning the dynamic behavior of the system. Model-based simulation and statistical analysis tools as provided by INCHRON greatly help to detect possible real-time requirement violations, and furthermore offer guidance in order to adjust and optimize an existing system configuration. Eventually, they help to document and prove[4] the feasibility of the dynamic architecture design or subsequently proposed design modifications.

[4]Depending on the safety and integrity level (SIL) standards like IEC 61508 or ISO 26262 for automotive require or at least recommend the usage of analysis and simulation tools for the verification of the dynamic architecture.

Another goal of the case study was to explore possible options for the integration of the different tools used in this project, e.g. for modeling the system architecture, compiling the AUTOSAR configuration, performing the timing simulation, and debugging and tracing on the target microcontroller. Although, some tool-integrations already exist, for example the INCHRON Tool-Suite can generate a model out of an AUTOSAR configuration in .arxml format, not all the relevant information is represented adequately in each model. In some case, we used the Python programing language and the model API of the INCHRON Tool-Suite to automatically generate the simulation model, and to extract execution times from a measured trace in order to update the parameters of the simulation model. In other cases, e.g. for the modeling of event chains or real-time requirements, the transformation from the system architecture model into the simulation model was done manually, mainly due to issues with the interpretation of the data flow semantics discussed in section IV-D of this paper.

In the future, we intend to increase the efficiency of the integration between the simulation and system modeling tools by extending the UML/SysML meta-model profile with the semantics defined in the MCSE methodology. This will make it possible, to automatically generate the event chain and requirement definitions used by the simulation, directly from the system architecture model, and thus eliminate the need to re-model them manually. This solution would also use the existing model API of the simulation tool, and can be maintained without high effort.

After that, we also want to evaluate, if formal verification methods as described in [6] can be applied in a real project environment, in order to find possible inconsistencies concerning the data and control flow already in the structural architecture model, and thus reducing the number of required simulation iterations for the optimization of the system. Finally, we plan to migrate the case study to a multi-core platform, in order to analyze and verify our assumptions for multi-core architectures.

References

[1] S. Anssi, S. T. Piergiovanni, S. Kuntz, S. Gérard, and F. Terrier. Enabling scheduling analysis for AUTOSAR systems. In *ISORC*, pages 152–159. IEEE Computer Society, 2011.

[2] Neil C. Audsley. Deadline monotonic scheduling, 1990.

[3] AUTOSAR Development Partnership. http://www.autosar.org.

[4] AUTOSAR Development Partnership. Specification of Timing Extensions, Final Version, Release 4.2.1.

[5] J. Belz, T. Kramer, and R. Münzenberger. Functional safety: Predictable reactions in real-time. *EE Times Europe*, 2011.

[6] Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, Mamoun Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang, and François Vernadat. Fiacre: an intermediate language for model verification in the topcased environment. In *ERTS 2008*, 2008.

[7] A. Burns. Scheduling hard real-time systems: a review. *Software Engineering Journal*, 6(3):116–128, May 1991.

[8] Jean Paul Calvez, Dominique Heller, and Olivier Pasquier. Uninterpreted co-simulation for performance evaluation of hw/sw systems. In *Hardware/Software Co-Design, 1996.(Codes/CASHE'96), Proceedings., Fourth International Workshop on*, pages 132–139. IEEE, 1996.

[9] J. Cooling. Rate monotonic analysis.

[10] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.

[11] C. Ekelin, A. Hamann, D. Karlsson, U. Kiffmeier, S. Kuntz, O. Ljungkrantz, M. Özhan, and S. Quinton. TIMMO-2-USE Methodology Description V2. Technical report, October 2012.

[12] Software engineering and Universidad de Cantabria real-time group. Modeling and analysis suite for real-time applications (MAST). http://mast.unican.es.

[13] C. Ferdinand. Worst case execution time prediction by static program analysis. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 125–, April 2004.

[14] INCHRON GmbH. INCHRON Tool-Suite. http://www.inchron.com.

[15] P.-E. Hladik, A. Déplanche, S. Faucou, and Y. Trinquet. Adequacy between autosar os specification and real-time scheduling theory. In *Industrial Embedded Systems, 2007. SIES'07. International Symposium on*, pages 225–233. IEEE, 2007.

[16] O. Kermia. Optimizing distributed real-time embedded system handling dependence and several strict periodicity constraints. *Advances in Operations Research*, 2011, 2011.

[17] Frédéric Leens. An introduction to i 2 c and spi protocols. *Instrumentation & Measurement Magazine, IEEE*, 12(1):8–13, 2009.

[18] Joseph Y.-T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.

[19] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[20] OMG. SysML Standard v1.2. http://www.omgsysml.org/.

[21] J. C. Palencia and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '98, pages 26–, Washington, DC, USA, 1998. IEEE Computer Society.

[22] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, November 2004.

[23] F. Simonot-Lion. Automotive Embedded Systems - The Emergence of Standards. In *15th IEEE International Conference on Emerging Technology and Factory Automation (ETFA 2010)*, Bilbao, Spain, September 2010. IEEE.

Qualitative simulation and validation of complex hybrid systems

Jean-Pierre GALLOIS and Jean-Yves PIERRON

CEA, LIST, Laboratoire d'Ingénierie Dirigée par les Modèles pour les Systèmes Embarqués

Point Courrier 174, Gif-sur-Yvette, 91191, France (e-mail: jean-pierre.gallois@cea.fr, jean-yves.pierron@cea.fr)

Abstract

Complex industrial systems need extensive validation and verification. Methods for this are well advanced in case of discrete systems. However, for hybrid systems that combine discrete and continuous aspects, they are not as well developed. To deal with this, qualitative simulation can be used, based on the principle of discretization by identifying domains of variation of continuous variables and tracking the evolution of these variables. A system can be discretized by representing its continuous parts, which are described by differential equations. When these are coupled with the discrete parts of the system, a fully discrete global model is obtained, on which formal techniques can be applied for the validation process. If the differential equations cannot be expressed clearly, it is necessary to establish a qualitative model describing the laws of evolution of continuous variables. We defined and tested a novel methodology that represents variations of continuous variables and the causal links between them to obtain mappings of system behaviors that are suitable for validation.

Keywords: hybrid systems, formal methods, qualitative simulation, validation

1. Introduction

Industrial systems are becoming increasingly more complex, requiring more powerful formal methods of verification and validation. Large models are usually difficult to solve analytically, so engineers use numerical simulations to study their behavior. But numerical simulation has limitations. It requires determining the values of parameters and initial conditions. In practice, these are often difficult to specify. Another difficulty is needlessly calculating the behavior of a system for parameter values that are of no practical interest in a lot of situations. More often it is of greatest interest to determine what class of behaviors is to be expected for a given constraint imposed on the initial conditions and parameters. That is why qualitative models and simulations have been used; they can cope with imprecise knowledge and with infinite numerical value domains by compacting them into abstractions, and using qualitative predictions to predict the significant qualitative classes of system behaviors. This type of qualitative reasoning is also suitable for process analyses and verifications.

In this paper we describe an approach proposed by CEA LIST, which uses formal techniques in order to automatically prove safety properties and to automatically generate a set of corresponding test scenarios. To that end, a methodology and a corresponding tool (named DIVERSITY) based on symbolic execution were developed at CEA LIST. This technique allows generating symbolic scenarios corresponding to classes of system behaviors, which are sequences of well-defined actions. Once the set of all possible symbolic scenarios is computed, it is possible to prove properties on this set and to generate concrete numerical tests from them (a single numerical test is sufficient to represent a given symbolic scenario). Symbolic execution is a proven way of overcoming both combinatorial explosion and computational redundancies. However, to analyze complex systems that include continuous components that are classically modeled by hybrid automata, it was necessary to extend the simulation method used in the DIVERSITY tool to support qualitative methods.

Qualitative reasoning has been applied in different domains as Artificial Intelligence, mathematics, economy, and much in bioinformatics this last decade [MGCL07]. Recent developments of formal tools and the increasing power of computers lead us to believe that these methods can be applied effectively to industrial-size hybrid systems.

State of the art

Hybrid systems can be described by a finite set of continuous processes and transitions, and a set of real and discrete variables on which they operate. The processes are generally defined by differential equations that give changes in the real variables. Transitions are discrete assignments and guarded with variables.

There are several classes of hybrid systems: if the differential equations, guards and assignments are linear, then the hybrid automaton is said to be polyhedral (polyhedral hybrid automaton). If the differential equations are equalities and inequalities that include constant parameters, then it is called rectangular (rectangular hybrid automaton). Finally, if the differential equations are equalities and inequalities that include the unit of time as a parameter, the system is said to be timed (timed automaton) and real variables are clocks (clocks). Timed automata are specially handled by the Uppaal [Be02] and Kronos [Bo98] tools.

For other types of hybrid systems, we can mention the following tools and methods, which are used to calculate a set of accessible states approximated by a superset of those states: Hytech [HHT97] for rectangular hybrid automata with transitions with linear guards; Checkmate [CK03] for non-linear continuous evolutions but with linear guards, evolutions approximated by a polyhedral; the Tiwari & Khanna method [TK02] for polynomial continuous evolutions, where the system is abstracted as a discrete system whose guards are polynomial. This latter approach [TK02] was chosen in our work because it can manipulate polynomial constraints that are widely used in differential equations involved in embedded systems, especially in the mechanical domain. The QEPCAD tool was used to deal efficiently with constraints including polynomial expressions [Br03].

If the system is not clearly described by differential equations, a qualitative model must be established using specifications that describe the evolution of continuous variables. Many works have been published on this topic and a good survey of them has been presented in “Modèles et raisonnements qualitatifs” [TD03], and, prior to that, in “A Survey of Techniques and Applications” [Qr95].

Technical context

A hybrid automaton [H96] is defined by a set of states and discrete and continuous variables. A transition between states has a guard that depends on the values of the variables and will assign new values to the variables corresponding to the new target state. For continuous variables, the laws of evolution can be described by more or less accurate differential equations in the context of the various possible formalisms and methods [TD03]. Continuous variables change according to these laws, representing the different states of the system. System states are timed in the sense that they have a duration. In contrast, the transition time between states of the system is assumed to be zero, as the system performs instantaneous assignments to variables.

To simulate this type of model, qualitative simulation [K86] [TK02], which is an alternative to numerical simulation is used. It is based on the principle of discretization by partitioning domains of variation of the continuous variables, based on the evolution trends of these variables (increasing, decreasing, or constant) indicated by the signs of their first derivatives (positive, negative or null). In this way, one can get a tree of abstract behaviors [RGLG03] that allows for coverage of the system states [GP14]. If these are coupled with the discrete part of the system, a fully discrete global model is obtained on which formal techniques can be applied.

If the differential equations are not available, it is necessary to establish a *qualitative model* that describes the laws of evolution of the continuous variables. One solution is to represent the velocity of variations for continuous variables and to establish causal links between them. This construction of automata modeling the evolution of continuous variables requires specific rules that we present here. These rules allow the definition of qualitative states and associated transitions where each qualitative state corresponds to a set of values for continuous variable with regards to its evolution law (corresponding to the sign of the derivative: positive, negative or null). Each transition defines a rule that allows the evolution to move from one set to another (e.g., move from a zero to a positive derivative).

2. Models with differential equations

As part of our study we retain the framework of models with differential equations of the first order (i.e., with first derivatives only), with polynomial integer coefficients, but to any degree and with any number of variables (this is the framework chosen by Tiwari [TK02] for qualitative simulation). Higher-order differential equations can be reduced to first-order differential equations by changing variables.

In summary, continuous variables change continuously with time and they can be calculated by the differential equations applied to the control states which define the laws of change. When a guard of a transition whose source state is the current state of control becomes true, this transition is fired, resulting in new values being assigned to certain variables. In the new state of control, which is the target state of the transition, the new differential equations applying to that state provide the corresponding laws for the further evolution of the continuous variables.

Qualitative simulation with differential equations

In this section, the context is assumed to be a continuous system described solely by differential equations.

Qualitative simulation, whose core principles were well defined in the 80s and which was refined in the 2000s, is based on the principle of discretization by partitioning areas of variation of the continuous variables of the system, based on their evolution trends (increasing, decreasing or constant) based on the signs of their first derivatives (positive, negative or zero). When all the discrete states corresponding to this qualitative partitioning are created, we can apply a simple algorithm based on the possible evolution of each variable. That is, a derivative may change its sign only when vanishing, because the process is continuous (for example a positive derivative must first be zero before becoming negative). This limits possible evolutions and thus reduces the size of the corresponding graph. Finally, the differential equation system allows generation of a transition system whose states are based on partitioning continuous variables into change in areas as described above and whose transitions are possible evolutions of these states. It has been shown that the result of such a qualitative simulation is an abstraction of the original differential equation system.

Symbolic execution

In the following, we describe the classical technique of symbolic execution used in the DIVERSITY tool as it is a based on the method presented in this paper, with an adaptation to hybrid automata context.

Starting from a transition system whose transitions are labeled by conditions and assignments to variables, it is possible to produce its corresponding symbolic execution tree, in which variables are described by expressions with parameters.

Given the current state S of the transition system, and a condition which is associated with the current state, the path condition PC, a symbolic execution constructs the tree of sequences of symbolic states, based on the following rules:

The root of the tree is the initial state (the current state at the beginning of the process);

For each possible successor of the current state S, a path can be constructed, if and only if the condition C on the parameters that makes a state S’ a successor of the current state can be true; in that case, an edge is constructed from S to S’ and the new symbolic values of variables are evaluated, applying the assignments, their symbolic values having been updated by substitution and simplified by a simplification procedure (for instance, if x + y is assigned to x, with a and b as initial values of x and y, then the final value of x will be a + b);

S’ is associated with a new constraint PC’, which is the conjunction of PC and C (if the PC is evaluated to false by a constraint solver then the path is cut);

The process is repeated with the successors of S’;

If a new symbolic state has already been reached along the same path (or in a previously calculated path if a more compact tree is required), then the execution of this path stops: this can be determining by comparing the appropriate sets of variables, by the criteria of inclusion (or equality if more precise calculus is required), for instance if the PC is $x>0$ in the previous state, and PC’ is $x >1$ in the final state, then the set of possible values of x in the final state is included in the previous state and the path can be cut;

The symbolic execution is over when all the possible paths have been assessed.

The discretization process

The discretization process requires creating a state machine for each variable representing an abstraction of its evolution. A variable x can be increasing, decreasing or constant, which is equivalent to the differential equations $dx/dt > 0$, $dx/dt < 0$, $dx/dt = 0$. This will be represented by a state machine with 3 states and the possible transitions between them corresponding to possible evolutions of the system.

Each transition is defined by an initial state, a guard, a final state, and a constraint expressed by a formula that must be evaluated a posteriori and whose evaluation will be based on each variable value after having been updated by its own state machine. The update of the variable depends on its direction of change (e.g., the new value of x > old value of x if $dx/dt > 0$ ...). As the constraints represent the differential equations according to the direction of evolution of the synchronized variables, the conjunction of these constraints represents the overall constraint that must be satisfied if the overall step of simulation is possible. To evaluate this overall formula the tool QEPCAD was used, which handles polynomial constraints. If the formula evaluates to false, the corresponding branch in the execution tree is cut. This process is repeated until the entire execution tree has been computed. As control states correspond to the direction of variation of each variable, we obtain for 3 states for 1 variable, and 3n states for n variables. As the set of states is finite, the process will necessarily terminate. This number of states (3n) gives a good idea of the potential complexity of the resulting models. It is clear that, in the worst case, the number of paths may be excessive. However, we assume that, if the method is applied to models of real-world systems, the number of classes of behaviors represented in this way will generally not be too large.

Complete system

In the case of a "traditional" hybrid system, the differential equation system which describes the continuous part of the system is combined with a discrete system in which there are discrete variables and transitions that affect these variables. For the qualitative simulation the discrete part of the system is unaffected by the discretization method described above. Control statements are for discrete variables "simple" states, while for continuous variables they are continuous process. The latter can be discretized with the qualitative simulation mechanism described above. Consequently, the entire resulting system is discrete, which means that it can be treated using traditional mechanisms (verification, test generation) that are applicable to discrete systems.

Nevertheless, there is a case which can be problematic: it occurs when the discrete part of the system contains variables whose domains of definition are infinite. Indeed, the number of states of the system in that case is potentially infinite, limiting the ability to fully analyze the system. In practice, if one is faced with such systems, several techniques are possible. We have chosen symbolic execution, which provides a representation that is either equivalent to the real system or a valid abstraction of it. It has been shown that when symbolic execution is applied, if the cut-off criterion is the equality of domains, then the execution tree is bi-similar to the digital implementation of the numerical tree. On the other hand, if the cut-off criterion is the inclusion of domains, then the symbolic execution tree is an abstraction of the numerical implementation of the system tree.

Working with Booleans or integers in the scope of Presburger arithmetic ensures the decidability of the process of the equality or inclusion. In this context, applying the symbolic execution to the discrete part of the system produces at least an abstraction of the execution tree of the system. As qualitative simulation also produces an abstraction of differential equations that it processes, the product of qualitative simulation and symbolic execution gives an abstraction of the execution tree of the system.

If we place ourselves in a less constrained framework, for example in the context of handling complex calculations of real numbers, we cannot apply the preceding cut-off criteria for the calculation of the symbolic execution tree (equal criterion or inclusion). In that case, the cut-off criterion chosen is generally a criterion of depth in the execution tree. In this case, we know that to this depth the symbolic execution tree is bi-similar to the numerical execution tree. It is thus necessarily an abstraction, and, therefore, for a chosen depth, we have as in the previous case of Presburger arithmetic, the ability to produce an abstraction of the numerical tree of the system.

As a result, we have an original method and an associated tool – based on symbolic execution for discrete parts of models and qualitative simulation for continuous parts – which can provide a usable abstraction of models of complex systems in every case. This approach reduces path redundancies to a minimum and, thereby, provides efficient support for analysis and verification of system properties.

Implementation

Qualitative simulation produces a discrete graph that is an abstraction of the system of the represented differential equations. The discrete graph is implemented by a transition system expressed in XLIA, which is the internal language of DIVERSITY.

Similarly, the discrete parts of the system can be translated from a model expressed in an industry-standard language like UML, SDL, or Matlab/Simulink, into XLIA, to be processed by the DIVERSITY tool.

Therefore, it is planned to jointly run the two systems (discretized differential equations and discrete system) translated into XLIA in a model by creating a model that includes these two XLIA models along with the connections between them. In the preceding discussion, it was assumed that the system has just a single differential part; but one can generalize this reasoning to a system with multiple differential parts, each of which can be discretized using the same process and then translated to XLIA.

An example: The Brusselator

The Brusselator is an example of an oscillating autocatalytic chemical reaction [AH03]. Its mechanism is given by the four following chemical equations:

1. $A \rightarrow X$
2. $2X + Y \rightarrow 3X$
3. $B + X \rightarrow Y + C$
4. $X \rightarrow D$

The dynamics of this system are described using differential equations and are typically resolved by a Jacobian.

Our goal is to provide a discrete description of the dynamics of the Brusselator. For this we divide the plane (X, Y) in areas according to increase/decrease X and Y, wherein these domains are synthesized by abstract states.

We thus obtain the graph of abstract states of X and Y, and therefore the behavior of the Brusselator system.

Equations

$Vx = 1 - (b + 1) X + a X^2 Y$ (the growth rate of the concentration of X)

$Vy = b X – X^2 Y$ (the growth rate of the concentration of Y)

Note 1: variables X, Y, and parameters a and b are positive.

Note 2: If the speed Vx is positive [resp. negative], the concentration of X increases [resp. decreases].

Abstract space

Division of space:

- 3 possible states for Vx (Vx = 0, Vx> 0 or Vx <0);

- 3 possible states for Vy (Vy = 0, Vy > 0 or Vy <0).

This makes a total of nine states (3x3):

State S1: Vx < 0 and Vy > 0

State S2: Vx = 0 and Vy > 0

State S3: Vx > 0 and Vy > 0

State S4: Vx > 0 and Vy = 0

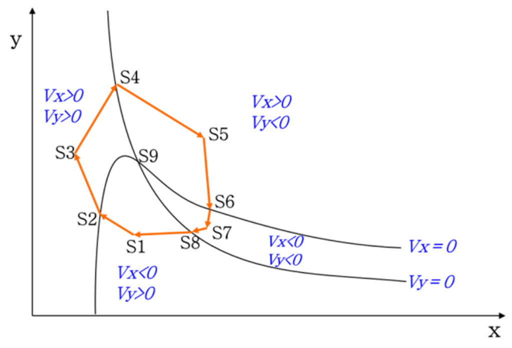
State S5: Vx > 0 and Vy < 0

State S6: Vx = 0 and Vy < 0

State S7: Vx < 0 and Vy < 0

State S8: Vx < 0 and Vy = 0

State S9: Vx = 0 and Vy = 0



The diagram represents the qualitative simulation of the Brusselator.

Dynamic analysis

Calculating all the possibilities of movement is equivalent to firing all transitions from each abstract state of the abstract space listed above. The simulation then yields the tree of all possible paths, which captures all possible dynamic behaviors.

Qualitative simulation shows strictly and explicitly the loop operation, which is characteristic of the Brusselator, without having to unfold the standard calculations of numerical analysis (by the Jacobian). On the other hand, the resulting abstract model provides an excellent support for the proof (e.g., it can show a loop clearly, or evaluate the intersection of the set of qualitative states with a forbidden region), especially for temporal logic properties. Finally, the abstract model can also help in the definition of test sequence (in this case, it demonstrates a characteristic sequence of the main loop by an experiment).

3. Models without differential equations

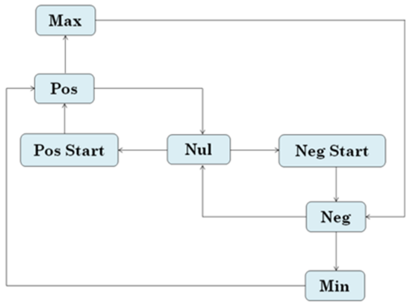
The method can be applied to a system including the physical data subsystem that is continuous. For this type of model the evolution of continuous variables laws can be described by differential equations or by abstract equivalents based on the tables of variation. In this section, we focus on the second case. The objective is to realize a hybrid automata representation of the system, and to apply to this model the technique of qualitative simulation that provides classes of behavior. The value of having such classes is to gain generality when validating the simulated model. Thus, if a loop is detected, it can be deduced that this loop exists for all the corresponding numerical paths. A major application of the qualitative method is getting a "mapping" of the behavior of the system. This mapping is intended to serve as a support for other purposes, such as co-simulation, HMI validation, etc.

Modeling the hybrid system

The inputs to the tool are specifications (which may be textual) and known data of the system, including engineering experiences. The model can then be encoded directly into XLIA, the internal language of DIVERSITY, in the form of concurrent state machines. It may be encoded using an industry-standard such as

SysML (an objective for our future work), but it is necessary to translate this to XLIA to use the DIVERSITY tool.

When the differential equations of a system are not available to describe the behaviors of the continuous variables, their laws must be expressed by hybrid automata, which must conform to specific methodological construction rules. These rules are summed up by the following diagram, which represents the possible sequences of qualitative states.



Indeed, they may be to the Nul state, their values are constant. They can begin to grow, which corresponds to the Pos Start state, or begin to decline, which corresponds to the Neg Start state. They may continue to decline, which corresponds to the Neg state, or continue to grow, that is to say the Pos state, eventually reaching either the Max or the Min state.

From the XLIA model DIVERSITY generates a tree of behaviors that can be represented in a graphical format for easier comprehension. For example, system cycles can be highlighted in this way.

Thus execution traces resulting from DIVERSITY simulations of the model can be analyzed and compared with potential scenarios configured by the operator in the simulator. In addition, one can compare the execution traces produced by the numerical simulator to what is expected by the qualitative model, to establish compliance of the numerical simulation to the qualitative simulation. DIVERSITY includes a feature that can automatically generate a compliance verdict.

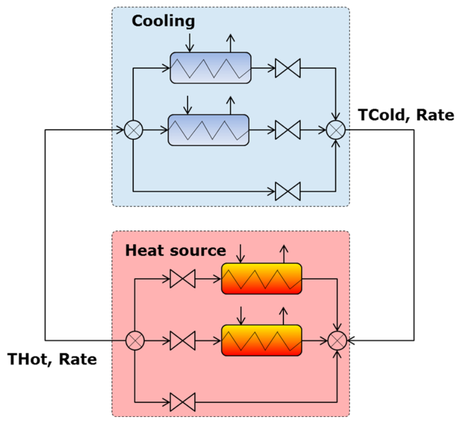
Outputs

DIVERSITY calculates a symbolic execution tree in which each execution path corresponds to a potential system behavior. If test is provided for each such path, a satisfactory coverage of behaviors can result.

The classical system properties that can be identified with this technique include detecting cycles (e.g., oscillations) and reachability (or not) of specific states (critical, non-critical, etc.).

Case study

This technique was applied to the SRI temperature control model (Intermediate Cooling System [NDB14]), which was coded directly into the internal language of DIVERSITY. The SRI is a basic system found in French nuclear power plants, which is used to provide cooling of auxiliary secondary circuits connected to the turbine process. This is a device for cooling several hot sources through an interface with a cold source, controlled by exchangers. It is of reasonable size and is sufficiently independent without being too simple, permitting it to be used in “Cluster Connexion” (a French BGLE2 project, COntrôle commande Nucléaire Numérique pour l’EXport et la rénovatION)) with available project resources. It contains logical and analog parts, and is, therefore, a good candidate for our study, which was coded directly into the internal language of DIVERSITY using the modeling methodology described above.



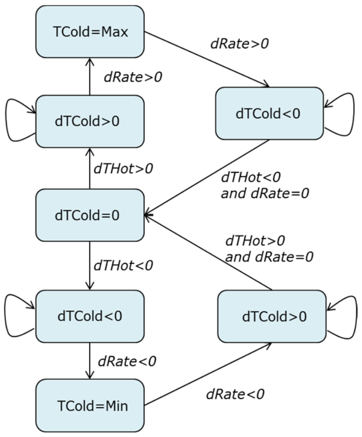
Three state variables in the SRI model were the primary subject of the simulation process study: the hot temperature of the circuit (input to exchangers), the cold temperature (output from exchangers) and the flow in the exchangers (given by the rate of the flow within an exchanger). All other parameters were assumed to be constant.

From the qualitative SRI model DIVERSITY generated a behaviors tree that could be represented in a graphical format for readability. This set of paths can highlight any potential oscillation in the control cycle, which can occur when the set temperature is exceeded, giving rise to several control cycles before stabilizing.

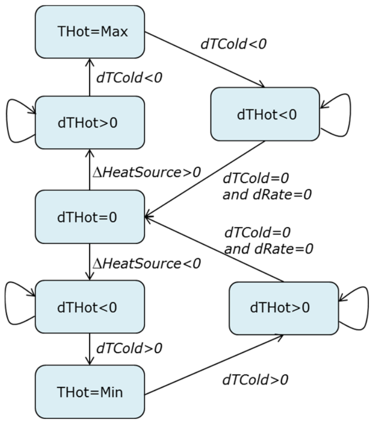
Another useful input to the DIVERSITY tool is a set of traces based on numerical simulation scenarios of the system. These traces capture observable variables in the numerical simulations. They can be used to demonstrate compliance of the numerical model with the XLIA model. Namely, the execution traces of the simulator can be compared to those of the qualitative model so that compliance of the numerical simulation with the qualitative simulation modulo these traces could be established. This was done by considering changes in temperature measurements in the circuit (by observing the oscillatory phenomenon of the regulation).

The following are graphical representations of the three main automata of the model that was realized in DIVERSITY and the graphical representation of the results of the qualitative simulation calculated by the tool:

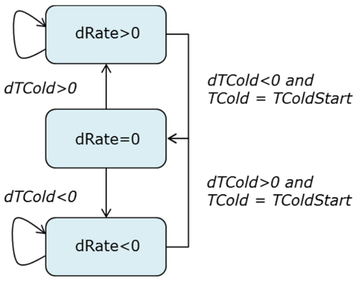
Cold Temperature: TCold.
Variation of TCold: dTCold



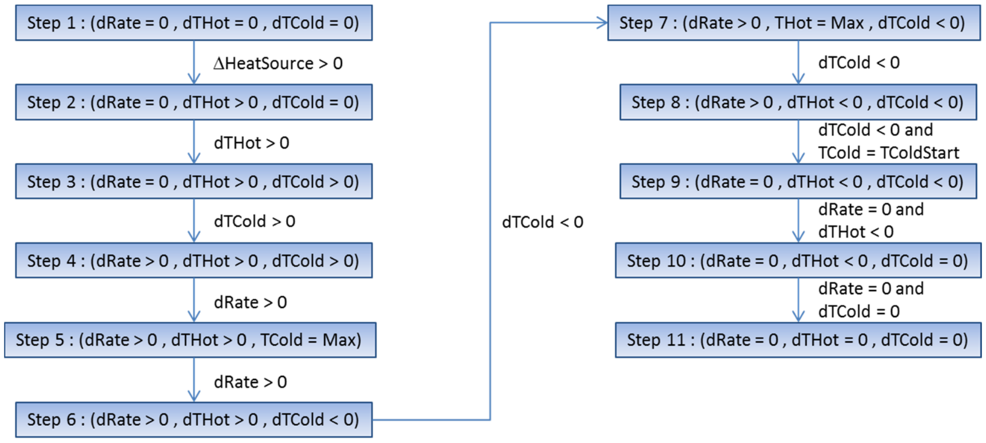
Hot Temperature: THot.
Variation of THot: dTHot



Exchange rate: Rate.
Variation of Rate: dRate



Note: ΔHeatSource>0 (resp. < 0) means an increasing (resp. decreasing) of heat source.
Thus we finally obtain a set of paths containing all potential behaviors. The following picture describes one of them.


Step 1 : (dRate = 0 , dTHot = 0 , dTCold = 0)
ΔHeatSource > 0
Step 2 : (dRate = 0 , dTHot > 0 , dTCold = 0)
dTHot > 0
Step 3 : (dRate = 0 , dTHot > 0 , dTCold > 0)
dTCold > 0
Step 4 : (dRate > 0 , dTHot > 0 , dTCold > 0)
dRate > 0
Step 5 : (dRate > 0 , dTHot > 0 , TCold = Max)
dRate > 0
Step 6 : (dRate > 0 , dTHot > 0 , dTCold < 0)
dTCold < 0
Step 7 : (dRate > 0 , THot = Max , dTCold < 0)
dTCold < 0
Step 8 : (dRate > 0 , dTHot < 0 , dTCold < 0)
dTCold < 0 and
TCold = TColdStart
Step 9 : (dRate = 0 , dTHot < 0 , dTCold < 0)
dRate = 0 and
dTHot < 0
Step 10 : (dRate = 0 , dTHot < 0 , dTCold = 0)
dRate = 0 and
dTCold = 0
Step 11 : (dRate = 0 , dTHot = 0 , dTCold = 0)

For this global case study we obtained a behaviors tree containing 49 states and 18 paths. Each path is a behavior cycle of the system. The longest path contains 19 states.

The complexity of the approach without using differential equation is similar to the complexity encountered with differential equations, in terms of numbers of states and potential paths of the produced tree. Based on that so we anticipate that, for reasonable case studies we will not have to model systems with very large numbers of distinct classes of paths, and that the method will remain practical. This will have to be evaluated more in the future with larger experiments.

4. Conclusion

The technique of qualitative simulation with differential equations has been evaluated using the CEA LIST DIVERSITY tool on several different models, including a model of chemical equations (the Brusselator), which is completely nonlinear. The main result of this was that a mapping of the behaviors of such systems can be calculated very quickly and presented in an abstract form, using qualitative states and transitions. For instance, in the chemical model example we obtained a single path which synthesized many concrete numerical paths.

Applying the method to the case when differential equations cannot be expressed clearly was also evaluated. A model of the cooling system in the French BGLE2 project “Cluster CONNEXION” was used. It provided interesting results, allowing the comparison of qualitative paths against the numerical paths obtained by numerical simulation, which are difficult to classify for validation activities. Since the qualitative paths are an abstraction of the numerical ones, they are more usable for formal activities and provide a more human-friendly and more suitable representations of behaviors (e.g., detecting oscillations or proving essential properties of the system).

Prospect

A tree produced by DIVERSITY can be used in co-simulation to validate other systems that interact with the hybrid system represented in this way. Indeed, co-simulation generally involves numerical simulators which often involve long computation times and which are necessarily configured only for specific scenarios, thus reducing the scope of exploration. In contrast, qualitative simulation provides a good abstraction of all system behaviors, which requires less computation time and thereby enables more exhaustive exploration.

We plan to construct a component FMU (Functional Mockup Unit) from the execution tree generated by DIVERSITY, which can play the role of observer and actuator, and which can be interfaced with a simulated system. This component can then be used to automatically drive a pre-computed execution scenario replacing the need for a human operator of the simulator.

Bibliography

[AH03] Ault, Shaun; Holmgreen, Erik. “Dynamics of the Brusselator” Academia.edu, 16 March 2003. 11/27/10 http://fordham.academia.edu/ShaunAult/Papers/83373/Dynamics_of_the_Brusselator

[Be02] Bengtsson, J., Griffioen, W. O. D., Kristoffersen, K. J., Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (2002). Automated verification of an audio-control protocol using UPPAAL. Journal of Logic and Algebraic Programming 52-3, 163-181.

[Bo98] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., and Yovine, S. (1998). Kronos: A model-checking tool for real-time systems. In Computer Aided Verification, LNCS 1427, 546-550, Springer.

[Br03] C.W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. SIGSA Bulletin, 37 (2003), pp. 97–108

[CK03] Chutinan, A., and Krogh, B. H. (2003). Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control 48, 64-75.

[Qr95] Qualitative reasoning: A survey of techniques and applications, P. Bourseau, K. Bousson, P. Dague, J.L. Dormoy, J.M. Evrard, F. Guerrin, L. Leyval, O. Lhomme, B. Lucas, A. Missier, J. Montmain, N. Piera, N. Rakoto Ravalontsalama, J.P. Steyer, M. Tomasena, L. Trave-Massuyes, M.R. Vescovi, S. Xanthakis, A. Yannou, AI Communications. Special Issue MQ&D: Qualitative Reasoning, Vol.8, N°3/4, pp.119-192, Sept-Dec 1995

[GP14] Jean-Pierre Gallois et Jean-Yves Pierron, INTERVAL, instanciation d'une plate-forme de validation pour les spécifications industrielles dans le cadre du projet CONNEXION, Génie Logiciel Hors-série « L'initiative Connexion : IDN et Contrôle-Commande »: 32-38, 2014.

[H96] T. Henzinger, The Theory of Hybrid Automata, Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96), pp. 278-292.

[HHT97] Henzinger, T. A., Ho, P. H., and Toi, H. W. (1997). HYTECH: A model checker for hybrid systems. STTT 1(1/2), 110-122.

[K86] KUIPERS B.J. (1986). Qualitative simulation. Artificial Intelligence vol 29 n3 239-388.

[MGCL07] Daniel Mateus, Jean-Pierre Gallois, Jean-Paul Comet and Pascale Le Gall, Symbolic modeling of genetic regulatory networks, Journal of bioinformatics and computational biology, Imperial College Press 05, 627 (2007)

[NDB14] Maxime Neyret, François-Xavier Dormoy et Jean-Christophe Blanchon, Méthodologie de validation des spécifications fonctionnelles du contrôle-commande, Application au cas d'étude du Système de Réfrigération Intermédiaire (SRI), Génie Logiciel Hors-série « L'initiative Connexion : IDN et Contrôle-Commande »: 12-25, 2014.

[RGLG03] Nicolas Rapin, Christophe Gaston, Arnault Lapitre, Jean-Pierre Gallois Behavioural Unfolding of Formal Specifications Based on Communicating extended automata ATVA 2003

[TD03] Travé-Massuyès Louise, Dague Philippe, Modèles et raisonnements qualitatifs ; TraitéIC2, série Systèmes automatisés, ISBN : 9782746207448, 2003.

[TK02] Tiwari, A., and Khanna, G. (2002). Series of abstractions for hybrid automata. In Hybrid Systems: Computation and Control, LNCS 2289, 465-478, Springer.

Session 11

Virtual Platforms

Thursday 28th, 09:00 – Ariane 1

Xvisor VirtIO-CAN: Fast Virtualized CAN

Jimmy Durand Wesolowski[12], Aymen Boudguiga[2], Anup Patel[3],
Julien Viard de Galbert[1], Matthieu Donain[4], Witold Klaudel[5] and Guillaume Scigala[1]
[1]OpenWide, 23 Rue Daviel 75013–Paris, France, *firstName.lastName[-lastName2]@openwide.fr*
[2]IRT SystemX, 8 avenue de la Vauve 91120–Palaiseau, France, *firstName.lastName[-lastName2]@irt-systemx.fr*
[3]Individual Researcher, Bangalore, India, *anup@brainfault.org*
[4]PSA Peugeot Citroën, Route de Gisy 78140–Velizy-Villacoublay, France, *matthieu.donain@mpsa.com*
[5]Renault, 1 avenue du Golf 78288–Guyancourt, France, *witold.klaudel@renault.com*

***Abstract*—Nowadays, vehicles are embedding more and more electronics to support new functions such as driver monitoring, lane keeping and adaptive cruise control. However, adding electronics makes vehicles more expensive. Fortunately, virtualization, via a hypervisor, reduces the number of embedded chips in vehicle by running different guests, i.e. Operating Systems (OSes), offering several services on the same board.**

As the communication between embedded controllers is compulsory for vehicles to function, an optimized virtualization of the Controller Area Network (CAN) bus becomes mandatory. CAN bus virtualization is challenging as it has to tackle the CAN arbitration mechanism and to provide CAN frame broadcast in a transparent manner. In this paper, we use the VirtIO virtualization interface with a virtual CAN service and framework to manage virtualized system external and internal CAN messaging.

***Index Terms*—Embedded Systems, Controller Area Network, Virtualization, VirtIO, Xvisor**

I. Introduction

By the end of the last century, transportation systems and especially vehicles replaced some of their mechanical functions by electronically controlled applications. Such applications include automatic control of windows, fuel injection supervision and automatic activation of car headlights. By the end of 2010, cars relied on software containing millions lines of code on 70 to 100 microcontrollers, namely *Electronic Control Units* (ECUs) [1], [2]. Nowadays, the customers needs for new Advanced Driver Assistance Systems (ADAS) services are increasing and so is the total number of embedded ECUs in the vehicles [3]. In fact, proposing new services for drivers on the road implies embedding more electronic boards, more communication buses, more cables and more software, ending up by increasing vehicle complexity. Consequently, car price is rising with respect to the number of its embedded ECU while it has to stay competitive on the market in order to attract more customers.

One interesting solution to reduce the number of vehicle embedded ECUs consists in using *virtualization* via a *hypervisor*. Virtualization allows to run different *virtual* automotive Operating Systems (OSes), called *guest OSes* or *guest systems*, simultaneously over one single physical board. Virtualization defines the hypervisor as a software layer (called *host*) between the *guest* OSes and the *real* hardware. The hypervisor emulates the hardware board for each *guest system* (Figure 1). In addition, it manages resources sharing between all the guests. However, virtualization comes with a major drawback which is computation overhead. As virtualization introduces the hypervisor layer between running guests and the board, it naturally induces a delay for accessing the hardware. That is, a virtual guest takes more time to access the memory, the network interfaces and the CPU than a classical OS running directly on the hardware. In fact, every time a virtual guest needs to access a resource, it has to pass through the hypervisor which introduces some overhead. Reducing this computation overhead is really important especially for communication scenarios. In practice, vehicle ECU exchange real time data frames through the Controller Area Network (CAN) bus [4]. When virtualization is used, CAN bus virtualization overhead must stay as low as possible.

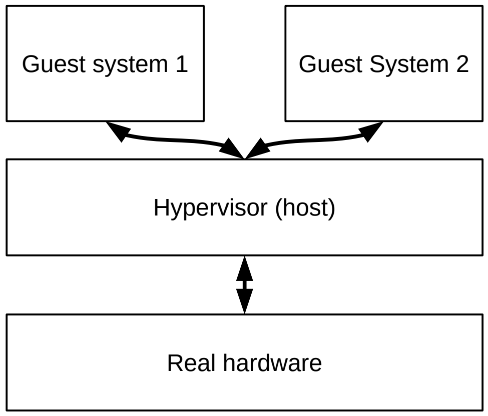


Fig. 1. Bare metal hypervisor

In this paper, we introduce an optimized virtualized CAN device. We use VirtIO API [5][6] to implement it for our hypervisor namely Xvisor [7].

The remainder of this paper is organized as follows. Section II reviews the CAN specification and introduces VirtIO and Xvisor. Section III depicts our solution. Section IV concludes the paper.

II. State of The Art

In this section, we give a brief description of the CAN protocol. Then, we introduce VirtIO and Xvisor.

A. Controller Area Network

The Controller Area Network (CAN) connects Electronic Control Units (ECUs) via a broadcast bus [4]. Each ECU is in charge of controlling multiple actuators or sensors. In mid-range cars, ECUs are used for Adaptive Cruise Control (ACC), fuel injection supervision, anti-lock braking system and comfort services management. ECUs communicate by exchanging CAN frames. These frames have one of the following types:

- *Data* frames exchange data between ECUs.
- *Remote* frames request the transmission of a specific *Data* frame.
- *Error* frames indicate the presence of an error over the CAN bus.
- *Overload* frames add an extra delay before the next frame transmission.

CAN frames do not contain information about their sender (i.e. source) or receiver (i.e. destination). They do not rely on interfaces addressing. In practice, each ECU manages a limited set of unique and distinct frame *identifiers*. A frame identifier defines an action that must be taken by the frame receivers.

CAN bits are encoded using a Non-Return to Zero (NRZ) code where 0-bits and 1-bits are encoded with different non-null voltage. The value of each transmitted bit is sampled at the end of a *nominal bit time* i.e. a bit time slot. In practice, CAN 0-bits are generally referenced as *dominant* bits and 1-bits as *recessive* bits.

CAN relies on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with a *bitwise arbitration* as bus access method. The bitwise arbitration concerns the value of frame identifiers. When two ECUs start the delivery of two different frames at the same time, the ECU sending the frame with the greatest identifier value stops its transmission at the reception of a dominant bit while it is transmitting a recessive bit.

B. Virtual Input Output

Virtual Input Output (VirtIO), is a virtualization abstraction API. It was created by Paul “Rusty” Russel to create a common layer for most virtual devices. As such, it avoids the proliferation of virtualization techniques in drivers, whose code and execution is similar to each others [8].

We choose to use VirtIO for CAN interface virtualization for three reasons. On the one side, VirtIO defines a clear and flexible API with well defined and optimized transport abstractions, that fit our needs. On the other side, VirtIO API is becoming the virtualization standard for host-guest communication interfaces [9]. Finally, it is recommended and supported by the LINUX community.

C. eXtensible Versatile hypervISOR

The eXtensible Versatile hypervISOR (Xvisor) is a *type-1 monolithic* embedded open-source hypervisor. A *type-1* hypervisor, also known as *bare metal hypervisor*, is a virtualization layer that runs directly on the hardware. Meanwhile *type-2* hypervisor is executed on top of an operating system. Xvisor is said to be *Monolithic*, as it has a common software for host hardware access, CPU virtualization, and guest IO emulation. Meanwhile, micro-kernelized software kernels contain basic hardware access and CPU virtualization, but they relies on a *management guest* to handle the other services (e.g. drivers, file systems, ...) [7].

By design, it has a fast interruption management. Benchmarks demonstrated a low overhead on both guest instruction execution and guest memory operations [10]. For example, a Xvisor guest yields an average of 2.550.336 Dhrystone Millions of Instruction Per Second (*DMIPS*), compared to 2.558.851 DMIPS for a native system. In addition, the Read-Modify-Write throughput on a guest is about 556,13 MB/s compared to 564,58 MB/s on a native system.

Xvisor allows having an emulated interface if needed, a paravirtualized one, or even a direct access to hardware when the resource is not shared.

To conclude, Xvisor flexibility suits well our need to use the already existing VirtIO interfaces, and to adapt it to create a new one for the CAN bus `VirtIO-CAN`.

III. The CAN VirtIO

In the following section, we first list the industrial contraints that must be fulfilled by our proposed solution, `VirtIO-CAN`. Then, we describe it in details.

A. VirtIO-CAN Prerequisites

The following list depicts the requirements regarding CAN virtual software:

1) The guest high level (userland) certified softwares must remain the same. Thus, the interface for the CAN must not change, i.e. it has to present a `SocketCAN` compatible driver for LINUX guests.
2) The different guest systems can communicate not only together, but also indistinguishably with the external physical bus.
3) The access must be controlled. Some systems must not have access to or read from the virtual and physical buses.
4) The message priority must be supported.
5) Even guest-to-guest messages must go through the physical bus, as those frames can be used for monitoring or debugging, for example, during a diagnosis with an On-Board Board Diagnostics plug.
6) Xvisor interface for the CAN management must be simple.
7) The overall performance must be comparable, if not indistinguishable, from a non-virtualized (or native) systems.

B. *VirtIO-CAN*

In order to reach good performances, the CAN communication cannot be fully emulated, i.e. a *fully virtualized*. Full virtualization implies trapping and handling every operation on the device from the virtualized system, in a complex manner, to simulate the hardware expected behavior.

As CAN resources are shared, direct hardware access, namely *passthrough*, cannot be used. This would give all guests the complete access to the same hardware and registers in particular. A guest could then overwrite the controller configuration or the data set by another guest. In addition, it may perform conflicting operations that cause errors. In the worst case, it may render the whole controller or system unsusable. Thus, direct hardware access can only be used for exclusive hardware access from one system, either the host or one guest.

Our considered solution, referred to by *paravirtualization*, uses a compromise between full virtualization and passthrough approaches. The guest driver provides only an interface on the host, with no knowledge of the underlying hardware. This results in a better overall performance and a lower complexity than emulation. Meanwhile, paravirtualization keeps the latter advantages of controlling the resource access.

We choose to achieve this communication through the VirtIO mechanism (presented in Section II-B). VirtIO defines a communication standard for the virtualization of multiple devices (console, network, disk, and soon graphical devices) over a variety of tranport medium (memory-mapped or PCI bus). It allows an identification of the device type, with the available features on the host and the guest side, and the configuration desired by the guest. High throughput data can be passed through VirtIO queues called *virtqueues*, and smaller specific data through the VirtIO *configuration read/write functions*.

VirtIO queues are designed for large data transfers. Their data footprint is at least 4 KB of memory with a queuing mechanism. CAN data frame has 29-bit long identifier and 64-bit long payload. Consequently, using VirtIO queues is inadequate because it introduces a large footprint and a cumbersome overhead. So, we decide to use the alternative data transmission mechanism: read/write configuration functions. That is, the guest access are done at specific offset on the host interface through VirtIO.

IV. VirtIO-CAN Implementation

In this section, we describe the implementation and mechanisms of the virtual CAN. First, we describe the initialization of the different layers. Then, we not only depict a frame transmission from a virtual system to the physical bus, but we also detail the reception process in the opposite direction. Finally, we present an internal frame transmission between guests on the same board.

A. *Initialization Phase*

1) Initializing the hypervisor driver: In order to avoid mixing the virtualization mechanism and the hardware management, the *hardware controller driver* is initialized and operates the same way without virtualization. Depending on the underlying material, CAN *mailbox* number varies. Note that a mailbox is a buffer designed to filter CAN frames with an identification mask. Each mailbox contains only one CAN frame. It is used as reception and emission buffer with a priority management. Mailboxes can be grouped to be used as First In First Out (FIFO) queues. Due to the design of the CAN bus, mailboxes assignment and setup is known and set in advance. We can define the priority of mailboxes and FIFOs. That is a FIFO contains a group of mailboxes with a defined set of priority.

FIFOs are set if the number of virtual-mailboxes is greater than the one available in the hardware controller. If the hardware does not allow setting FIFOs, the service will not be initialized to prevent misuse.

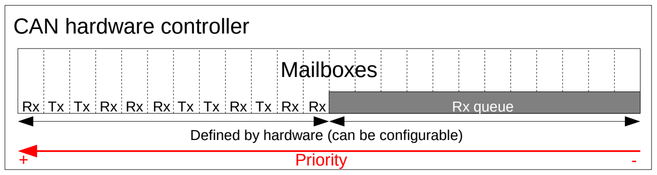

CAN hardware controller
Mailboxes
Rx Tx Tx Rx Rx Rx Tx Tx Rx Tx Rx Rx
Rx queue
Defined by hardware (can be configurable)
+
Priority
-

Fig. 2. CAN hardware controller system

2) Initializing the hypervisor service: We denote by `VirtCAN` the hypervisor service for the CAN framework. As such we avoid the ambiguity with the VCAN [11]. `VirtCAN` creates virtual-mailboxes with FIFO support to separate the hardware support from virtual system needs. It also provides a communication layer through the VirtIO framework to the guest system.

The host configuration sets the number of mailboxes and FIFOs to be created (Figure 3). The virtual-mailboxes are buffer slots of four 32-bit words each. The three first slots correspond to the CAN frame itself, and the last one serves for control and status storage (Figure 4).

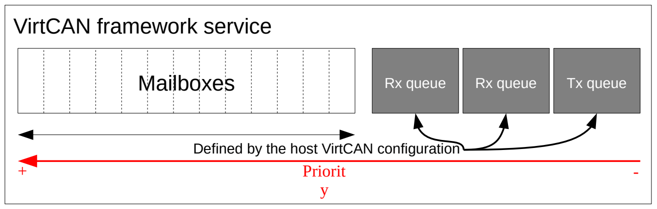

VirtCAN framework service
Mailboxes
Rx queue
Rx queue
Tx queue
Defined by the host VirtCAN configuration
+
Priorit
y
-

Fig. 3. `VirtCAN` system

Sending a frame on a particular mailbox returns an error to the guest if the latter already stores one that was not sent. A mailbox read twice without new matching frame also returns an error. However, by design a CAN messaging system avoids that.

The virtual FIFOs use the same buffer format (Figure 4), but frames are queued. A guest system can read or write a defined number of frames before an error is reported. In general, FIFOs are used for lower priority frame reception.

However, the `VirtCAN` also permits the creation of transmission queues.

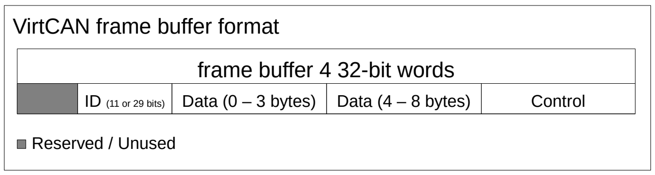


Fig. 4. `VirtCAN` framework buffer format

The hypervisor also has a configuration file for each guest to describe which and how a virtual-mailbox is used by this virtual system. Two guests cannot share the same virtual-mailbox to transmit or to receive a frame. However, they can have access to the same FIFOs. Each guest mailbox is a memory pointer to a virtual-mailbox.

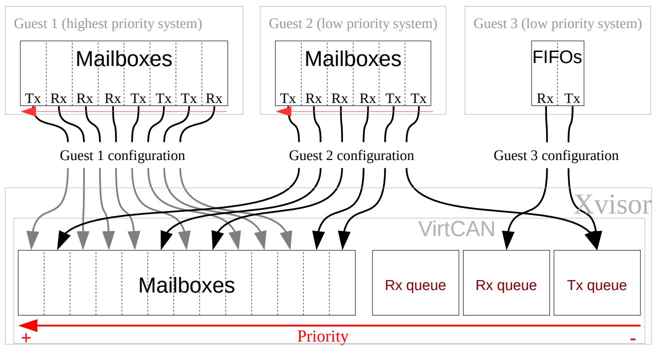


Fig. 5. Guest to `VirtCAN` mailbox association

3) Initializing the guest driver: The guest driver uses the VirtIO API to detect the VirtIO device and configure the required mailboxes and FIFOs. The hypervisor checks every guest setup to ensure its consistency.

B. Guest CAN Frame Transmission

Guests start transmitting at the end of the initialization phase. The VirtIO driver receives the frame from the higher layers (application, library, runnable, task,. . .). Then, it sends it to the hypervisor `VirtCAN` framework. The guest side VirtIO writes the frame at the offset formed by the addition of the mailbox or FIFO offset and an initial base offset for mailboxes. The latter is set by the `VirtIO-CAN` API.

The CAN frame identifier (`ID`) is written on a 32-bit word while the frame data are written on 8 bytes. Finally, the 32-bit control word is set. Note the frame payload can be less than the reserved 8 bytes. If a guest try to write a CAN frame with an unauthorized identifier, it is discarded and a warning is returned by the framework.

For example, to write a frame in its mailbox `n` , a guest has to write the 4 words of 32-bit in the VirtIO device at the offset: `mailbox offset + VirtCAN buffer size × n`

When the control word is written, the hypervisor `VirtCAN` service writes the frame to the CAN hardware controller. If virtual-mailboxes number exceeds the physical-mailboxes one, the excedent content is written in the highest priority transmit FIFO. The system integrator knows the number of excedent mailboxes as he knows the total number of the mailboxes of the guest, the `VirtCAN` and the hardware.

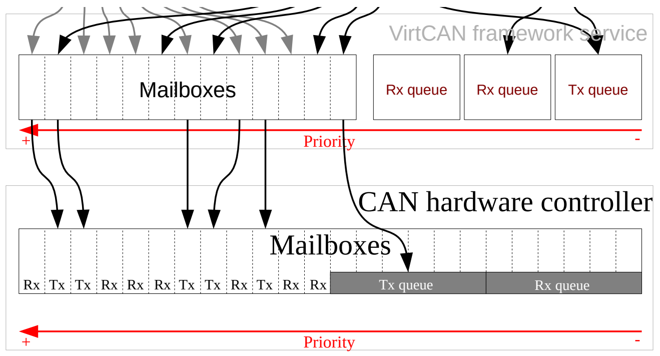


Fig. 6. `VirtCAN` mailbox to hardware frame transmission

C. Guest CAN Frame Reception

The frame reception process adapts the same idea of frame transmission of Section IV-B. When a frame is received by the hardware controller, it is copied in the virtual-mailbox matching the frame identifier. The number of physical-mailboxes can be lower than the total number of virtual-mailboxes. Then, the host must be able to receive all the frames. If no mailbox matches a frame identifier, it is queued in the highest priority FIFO. Note that the physical-mailboxes must be principally used for the higher priority frames. If all the reception FIFOs are full, or if there is no FIFO at all, the frame is discarded. In addition, an error message is returned.

Once a message is received in the `VirtCAN` layer, the service checks which guest is associated to this mailbox, write a *flag* corresponding to the guest mailbox number in the VirtIO configuration memory at a specified offset, and triggers a CAN interrupt to the guest.

When the guest driver receives the interrupt, it reads the VirtIO device for the mailbox flags. Then, the driver can submit the frame content to the upper layers. The reception flags synchronization between `VirtCAN` and the guest is not necessary. In fact, all VirtIO write operations done by the guest are trapped by the hypervisor. Then they are used by `VirtCAN` which updates itself the real flag status.

D. Guest-to-Guest CAN Frame Transmission

The Guest-to-Guest CAN frame transmission matches the aforementioned CAN frame transmission. The different layers operates as described in Section IV-B. In addition, we create a special association between a guest reception mailbox and a transmitting virtual-mailbox. When a frame is sent by a guest, these association sets are checked to notify the destination guest(s). Then, `VirtCAN` sends the frame to the hardware controller.

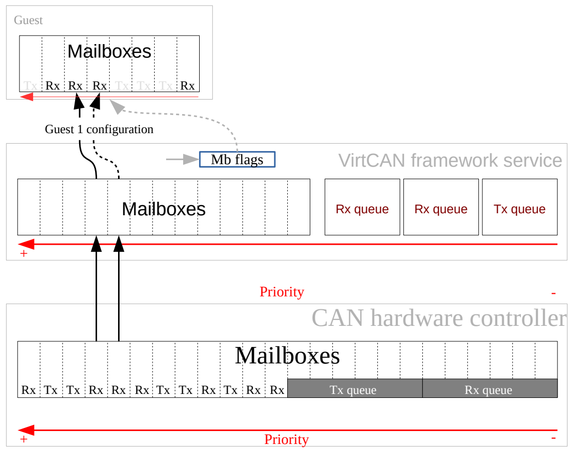


Fig. 7. Hardware frame reception to guest layers

If the frame is also destinated to another guest on the same board, VirtCAN set the guest flags, accordingly to the predefined association.

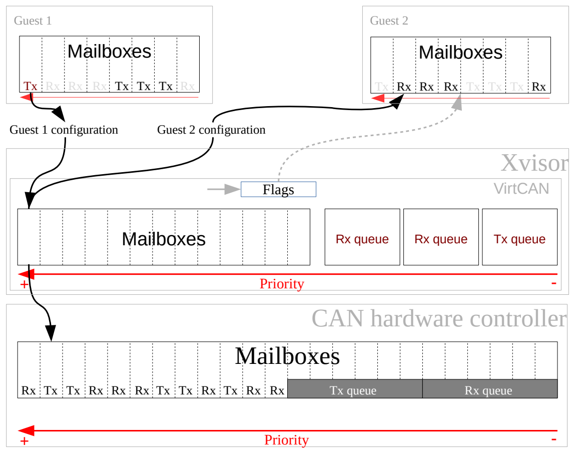


Fig. 8. Internal frame communication

V. Conclusions

In this work, we propose a framework, VirtCAN, to manage internal and external CAN bus virtualization with error management. The virtualization scheme is also preserved, as the hardware and the virtualization layers are clearly separated as seen in Figure 9.

As presented in the Xvisor memory benchmarks [10], the memory throughput of emulated operating systems is as good as a native one (up to 98%). There are only two memory copies for both frame direction: one from the guest to VirtCAN, and one from VirtCAN to the hardware for a frame emission or reception. Moreover, no specific scheduling is required to implement this mechanism.

Our future work consists in implementing VirtCAN. As the higher CAN transmission speed is up to 1 MB/s, and the memory throughput between guests and host is superior to 100 MB/s, we expect a low virtualization overhead, and a very close bus occupation and behavior between the native system and the virtualized one.

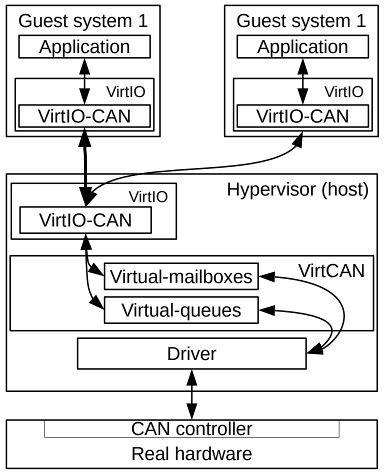


Fig. 9. System design

References

[1] Charette Robert. This car runs on code, 2009.
[2] J.Motavalli. The dozens of computers that make modern cars go (and stop). 2010. Accessed: 2015-06-16.
[3] G.Pitcher. Growing number of ecus forces new approach to cars electrical architecture. 2012. Accessed: 2015-06-16.
[4] CAN Specification Version 2.0. Standard, Bosch, Stuttgart, September 1991.
[5] Jake Edge. An api for virtual i/o: virtio. 2007. Accessed: 2015-10-23.
[6] Dor Laor. VirtIO, 2008.
[7] Anup Patel. Xvisor: eXtensible Versatile hypervISOR. Accessed: 2014-05-05.
[8] J.Edge. An api for virtual i/o: virtio. 2007. Accessed: 2015-06-17.
[9] R.Russel. virtio: Towards a de-facto standard for virtual i/o devices. pages 1–2, 2013. Accessed: 2015-04-08.
[10] A. Patel, M.Daftedar, M.Shalan, and M.W.El-Kharashi. Embedded hypervisor xvisor: A comparitive analysis. pages 4–9, 2015.
[11] Urs Thuermann. The virtual CAN driver.

An Experiment on Exploiting Virtual Platforms for the Development of Embedded Equipments

P. Cuenot1[3], E. Jenn1[4], E. Faure2, N. Broueilh2 , E. Rouland1[5],
1: IRT Saint-Exupéry, 118 route de Narbonne, 31042 Toulouse, France
2: ASTC France, 42 avenue du Général De Croute, 31100 Toulouse, France

Abstract: *Virtual engineering* methods and tools based on simulation have become a privileged mean to reduce time-to-market and product cost. However, design and verification activities still need to be improved to manage the ever increasing complexity of electronic products and their interactions with heterogeneous environments. In particular, an important challenge is to master the real time properties of the product composed of interacting hardware and software components.

In this paper we propose a pragmatic approach to use virtual platforms to verify gradually and accurately the properties of a system under design. We illustrate the approach on an example.

Keywords: Virtual platform, SystemC, Verification. Simulation, Heterogeneous environment.

1. Introduction and Motivation

New embedded electronic systems impose a new leap in product integration. The progress of the System on Chips (SoC) market share clearly illustrates this need. In parallel, embedded electronic systems or Electronic Control Units (ECU) need to be optimized to integrate new functions usually developed using analog parts at an acceptable cost.
Basically, the challenge is to maximize the usage of SoCs while keeping reasonnable development costs. Embedded electronic systems are usually reactive: they are part of some control loop closed through their environment. The temporal properties of the elements of this loop are of prime importance as they determine the overall performance of the implemented function (response time, stability, etc.). Accordingly, those properties must be carefully monitored and the product (including drivers, IPs, sensors, actuators…) shall be carefully designed to comply with those properties. In this context, being able to simulate a complete control chain within its environment is of major interest, and the benefits of the approach are as high as it can be applied soon, continuously and “smoothly” during the development process.

Towards this goal, there is a strong need to move from segregated hardware and software simulation to system-level simulation. This requires the modelling and simulation of the system’s components and environment at various levels of abstraction, representativeness, and accuracy. The SystemC language and simulation kernel [1] provide such capabilities.

SystemC is supported by an open source ecosystem organized in the context of Accelera Systems Initiative [1]. Recently, the Electronic Design Automation tools industry (EDA) extended commercial offers by integrating SystemC. The tool environment offers modelling services on the top the language, scripting capabilities to automate test execution, co-simulation interface to interact with external tools or actual electronic prototypes, etc.

Despite (or because of) the large offer of technical methods and tools, some methodological guidance is required to ensure a safe, high-quality, and cost-effective development and verification process.

Therefore, we (i) propose an iterative process to optimize the use of SystemC, (ii) apply it on part of a system, and (iii) show its benefits. Focus is placed on verification activities carried out with the system’s environment to demonstrate the preservation of the components properties during the successive development phases.

Our paper is structured as follows. Section 2 introduces the virtual platform concept and gives an overview of some significant related works. Section 3 presents the proposed approach for an iterative development and verification process. In section 4, the experimental setup is explained and section 5 gives some evaluation results. Finally, the conclusion reports feedback on methods and tools evaluation by application of the process and draws perspective for future work.

2. Related Work

A virtual platform is a hardware simulator executing embedded software. The hardware simulator is usually built on top of an Instruction Set Simulator (ISS) of the processor connected via buses to memory and peripherals such as timers, general I/O, communication interface, etc. The ISS may be implemented in SystemC or in any other general purpose language.

In a typical configuration, communications are modelled using SystemC Transactional Level Modelling

[3] Seconded from Continental Automotive France, 1 avenue Paul Ourliac, 31036 Toulouse France
[4] Seconded from Thales Avionics, 105 avenue du Général Eisenhower, 31100 Toulouse, France
[5] Seconded from ACTIA Automotive, 5 rue Jorge Semprun, 31400 Toulouse, France

(TLM) in order to achieve high simulation speed. Peripherals and memory are register accurate, they communicate according to the TLM paradigm, and their behaviour is implemented in SystemC/C++. The application software is the actual code compiled for the target processor. The SystemC non-preemptive simulation kernel orchestrates the execution of all components of the platform.

Among typical examples of virtual platforms, we can mention the Infineon Tricore™ SoC [2] based on the QEMU ISS or the SockROCKET LEON3 virtual platform developed by ESA [3] implemented in Python with SystemC/TLM buses and peripherals. Note that the ISS can be developed by silicon suppliers from proprietary architecture modelling languages, such as Freescale's ADL/uADL [4], and then be integrated with peripherals to build a complete platform.

Virtual platforms have been experimented on various industrial use cases, such as automotive power train applications [6]. Those experiments have demonstrated the need for appropriate methods during the development of the platform components and their integration.

In the industry, the different uses of virtual platforms map to the organisation of the supply chain: design and verification of SoC on the silicon suppliers side (component), design and verification of software on the equipment suppliers side (system).

On the system side, virtual platform are used to estimate the performances of hardware and software architectures, and perform early verification activities. In particular, debug and verification phases are simplified thanks to the high observability and controllability of virtual platforms compared to real hardware. Furthermore, using virtual platforms moves the simulated hardware parts out of the critical path: hardware development phases can start once the definition of the hardware has matured and has been (virtually) validated.

Finally, virtual platforms also provide:

- A high configurability and flexibility allowing a particular platform to be configured and elaborated within minutes *provided that the models are available*.
- A capability to integrate models with heterogeneous abstraction levels (in particular temporal abstractions thanks to the versatility of SystemC/SystemC-TLM/ SystemC-AMS)

In its Return On Investment (ROI) analysis on electronic system level design, Synopsys [5], one of the main EDA tool supplier, announces cost reduction opportunities by reducting the number silicon iterations and a productivity increase of x2-x5 for software development.

To reach such a ROI, the virtual platform design flow shall be optimized so as to (i) minimize the cost of models and (ii) maximize the credit one may take from verifications performed using those models. An approach consists to use simulation models as a form of an "executable design artifact" that is refined all along the development process, from the implementation agnostic logical level down to the physical software and hardware levels. Obviously, this approach is conditionned by the quality of the model and particularly on the properties preserved by the model. Those qualities must be clearly stated and become part of a contract binding the provider and the user of the model. A specific care shall be taken on timing properties of the hardware model and of the environment impacting the overall system behaviour.

The Socket project [7] has proposed such a design flow for the development of critical embedded SoCs. The development steps maps to the SystemC/TLM modelling styles. This allows to co-simulate hardware and software, and to introduce and verify properties gradually (in particular temporal properties). The top level design of the SoC architecture is focused on hardware bus traffic optimization, not on the complete set of properties allocated to the SoC. In this project, the modelling of the environment is limited since the systems considered are not reactive.

The COMPLEX project [8] uses UML/MARTE to model and explore the design space of embedded systems. Power and performance are the exploration criteria. The various abstractions of the processor micro-architecture, of the SoC internal buses, etc. allow an early and efficient simulation. Those abstractions also impair accuracy and raise sensitivity problems (in particular with target compiler optimization versus native compiler and internal SoC bus communication control). Compared to Socket, COMPLEX placed focus at system level where interconnected ECUs communicate through communication buses. The precision in low-level modelling is not considered besides the use of existing SystemC component libraries. Additionally the verification activities are not formalized.

With respect to the previous works, our approach is aimed at covering a larger modelling spectrum, from high level models down to behavioural hardware models. Emphasis is placed on verification of real-time properties considering the ECU's environment.

3. An Approach for the Development and Use of Virtual Platforms

To benefit the virtual platform's "good properties", the objective is to (i) minimize the development cost of the virtual platform and (ii) maximize the usage of the virtual platform.

To achieve the first objective, one shall (i.a) maximize the reuse of existing models (possibly reusing them from previous developments), (i.b) maximize the automatic generation of platform models (or skeletons) from existing design models, (i.c) develop simple models covering the necessary and sufficient features and details with respect to the validation and verification objectives. These topics will be addressed in Section 4.d.

To achieve the second objective, the strategy is two-pronged: (ii.a) the “most expensive” problems shall be identified and tackled first, (ii.b) the model shall be detailed up to the point where the necessary and sufficient precision and accuracy are obtained to solve (ii.a).

Here, we propose an informal approach somewhat similar to a Failure Mode and Effect Criticality Analysis (FMECA) applied on a development process. This approach takes into account: the cost of evaluations including the cost of model development, the cost of model execution and the risk inherent to a “sloppy” evaluation. According to this interpretation, the frequency of occurrence of an error is related to the quality of the measures (its accuracy and precision). The gravity integrates the potential impact of the error on the development process (e.g., the number and nature of the activities to be redone), and the impact on the product under design. Once the analysis is complete, reducing the overall risk consists to reduce the probability of occurrence of the evaluation error (e.g., by refining the model in order to account for phenomena that have a significant impact on the behaviour of the system), (ii) reduce the gravity of the error by improving the robustness of the process (e.g., by introducing margins), (iii) detect evaluation error by adding additional checks.

In this paper, focus is placed on the temporal properties, so the trade-off between cost and accuracy/precision is essentially focused on the modelling of temporal aspects. We consider the three levels — or programming styles —defined by SystemC-TLM: untimed (U or *functional*), loosely-timed (LT), and approximately-timed (AT). As the design improves, the verification objectives get more and more focused and may eventually require a fine grain temporal modelling. This change is determined by the property to be verified.

Failure modes

Contrary to hardware devices, there is no common, standardized list of failures at functional level. Candidate failure modes are identified and selected on the basis of expertize, experience, etc. Here, focus is placed on temporal errors such as under and over estimation of delays, non-compliance with SW/HW interaction sequencing or hardware design constrainst. The objective is to determine which verification technique to apply by considering the design faults, their effect on the system and the cost of the detection / mitigation means.

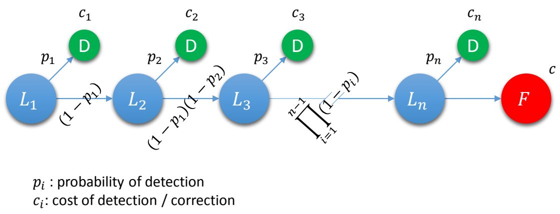
Criticality

Our estimation of criticality is (roughly) estimated by a cost. The principle is trivial: the cost of the verification means shall be somehow commensurate or related to the direct and indirect costs of the error. The cost of the error is roughly estimated by its criticality, which depends on its probability of occurrence and the costs of its effects including the cost of detection means (technical and human) and the cost of error elimination (redesign and refactoring,…).

Error propagation

With respect to a classical FMECA, the approach differs because design errors propagate across design artefacts (space) and across design phases (time). In particular, an error in the dimensioning of a parameter in phase i may impact the design choices done in phase $i + 1$, and so on.

Any verification approach is basically aimed at preventing such propagation by (i) detecting design errors as soon as possible in the design process and (ii) as soon as possible in the dependency chain that relate design artefacts.



c_1
c_2
c_3
c_n
D
D
D
D
p_1
p_2
p_3
p_n
c
L_1
$(1-p_1)$
L_2
$(1-p_1)(1-p_2)$
L_3
$\prod_{i=1}^{n-1}(1-p_i)$
L_n
F
p_i : probability of detection
c_i: cost of detection / correction

Figure 1: FMECA overview

This “approach” has been applied on some simple functions of our experimental setup. The corresponding use cases are described in section 4.b and 4.c.

Back to the important question of the ROI of such multi-viewpoint, multi-level approach, the following questions shall be answered:

- What is the cost of developing (all) those additional models? The answer obviously depends on their complexity. Hence, the development effort for a SystemC-TLM timed model of a simple timer is lower than one men×month, while a complex ISS can require more than twenty men×months.
- If models are develop by some third party, how do we formalize the “contract” that binds the model provider to the model user? Stated differently, how can we specify the domain to obtain significant results with the virtual platform model?
- Finally, how does this additional cost compares to the gain due to the early validation?

In the sequel of the paper, we propose to answer those questions by applying the virtual platform approach on a small example: an autonomous rover.

4. The experimental setup

In order to evaluate the proposed development process and supporting tools, and estimate its actual benefits, we use it for the development of a small mobile vehicule, or “rover”, named “TwIRTee”.

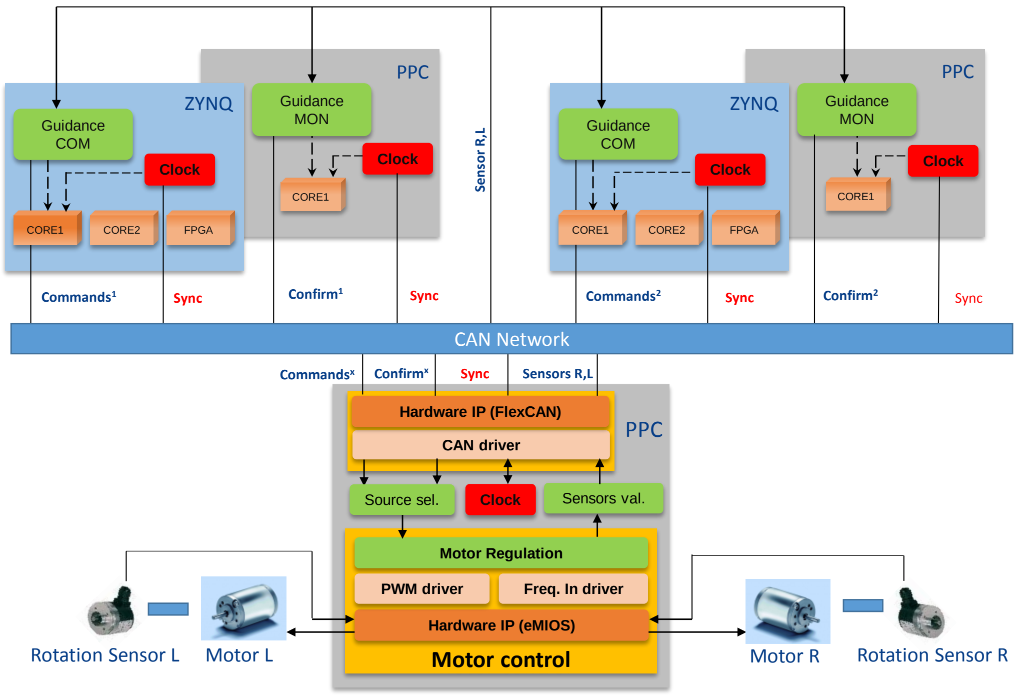


Figure 2: Overview of the TwIRTee equipment

a. The TwIRTee demonstrator

TwIRTee is a three-wheeled autonomous rover fitted with a camera and various other sensors (odometry, positioning,…). Its operational role is very simple: (i) move itself on some predefined tracks from a point A to a point B (a "mission") while avoiding other rovers.

TwIRTee is developed within the INGEQUIP project at the Toulouse *Institut de Recherche Technologique* (IRT) Saint-Exupéry. IRTs are new research structures established under the auspices of the French *Agence Nationale de la Recherche* (ANR) aimed at favouring the transfer of innovation from laboratories to industries.

TwIRTee is designed so as (i) to cover the major topics addressed in the project namely: early validation, architecture exploration, performance prediction, and formal verification. Furthermore, it is aimed at covering issues, functional and architectural elements specific to three industrial domains: automotive, space, aeronautics.

Accordingly, the missions, functions and the architectural elements are determined so as to tackle or exercise one or several issues: for instance, the "localization" function relies partially on imaging so as to exercise hardware / software space exploration and co-design; the highly redundant architecture provides the experimental setup to perform early performance evaluations (including dependability).

An overview of the TwIRTee plaftorm is given on Figure 2: the computing platform is composed of 2 COM/MON channels that host the main "mission" functions and one channel dedicated to power supply generation and motor control. A clock synchronization protocol is implemented and distributed on each ECU communicating by the CAN network.

The rover displacements is achieved by the motor control ECU controlling a two-wheeled powertrain composed of 2 CC motors, 2 reduction gearboxes, 2 quadrature encoders, 2 wheels. The setpoint for motor regulation is selected from the 2 commands (COM) and monitoring (MON) channels.

The methods introduced in Section 3 are applied on two simple functions: the clock synchronization (Fck) and the PWM motor control (F_{PWM}).

b. Clock synchronization

The clock synchronization protocol is used to provide the rover's computation units with a "common" time reference. The protocol has been proposed in [9]. It is built on top of the CAN network.

Principles

The common time reference – or virtual clock (vc) – satisfies the precision, rate and accuracy properties. The *precision* property states that no two synchronized virtual clocks may differ by more than a given value. For instance, at any time t, any virtual clock shall show a time "less than 100μs" from any other virtual clock. More formally, if nodes k and l participate to the protocole:

$$\exists\delta_v\colon |vc_k(t) - vc_l(t)| \leq \delta_v \qquad \text{(P1)}$$

In practice, the achievable precision depends on two main parameters Γ_{tight} and $\Gamma_{\text{max agree}}$:

$$\delta_v = \delta_{vi} + 2\rho T \qquad \text{(P2)}$$

$$\delta_{vi} \geq (1+\rho)\Gamma_{\text{tight}} + 2\rho\Gamma_{\text{max agree}} \qquad \text{(P3)}$$

where

- T is the resynchronization period, ρ is the physical clock drift
- Γ_{tight} is the network propagation delay (around 10μs), $\Gamma_{\text{max agree}}$ is the agreement delay which depends in particular on the number of tolerated faults and on the background traffic of higher priority.

First, let's analyse the failure modes, their "probabilities" of occurrence, and their effects.

Failures

- F_{F1M1}: Erroneous $\Gamma_{\max agree}$, underestimation
- F_{F1M2}: Erroneous $\Gamma_{\max agree}$, overestimation
- F_{F2M1}: Erroneous Γ_{tight}, underestimation
- F_{F2M2}: Erroneous Γ_{tight}, overestimation

Probabilities of occurrence

- O_{F1M1}: VERY HIGH, because (i) many factors contribute to the communication times, and (ii) the dynamic behaviour of CAN is not trivial (see [10] and [12]).
- O_{F1M2}: LOW, because the mode for fault F1 is much likely F1M1.
- O_{F2M1}: VERY LOW because this parameter is part of the specification.
- O_{F2M2}: VERY LOW *(*same reason as F2M1).

For space reason, we do not consider failure modes F2 any longer.

Effects and cost impact

- E_{F1M1}: Actual accuracy lower than expected. Cost impact is MODERATE: (i) the system being redundant (MASTER/SLAVE, COM/MON) many functions rely on the synchronization precision in particular via discrepancy margins, confirmation times are affected. However, the risk for a large effect on the synchronization is low thanks to the ρ[1] factor in (P2).
- E_{F1M2}: Actual accuracy greater than expected. Cost impact is LOW: the network and CPU loads are higher than necessary but no rework is expected. (Note that, generally speaking, the impact could be very high because it could lead to select and oversized computing platform and / or network. In our very case, there is no risk of such effect).

In the absence of dedicated detection means, F_{F1M1} and F_{F1M2} are likely to be detected only in operations.

Probability of detection

- D_{F1M1}: VERY LOW because $\Gamma_{\max}$ is a worst-case situation that is difficult to observe in operation.
- D_{F1M2}: VERY LOW because the effects are hardly visible.

Cost of correction

- C_{F1M1}: VERY HIGH
- C_{F1M2}: VERY HIGH

[1] ρ is in the range of 10^{-6} s/s

From the combination of a MODERATE cost, a VERY LOW detection probability and a VERY HIGH detection / correction cost, we decided that it was worth adding a new detection means with a MODERATE cost and HIGH detection coverage.

Consequently, we decided to test the clock synchronization protocol on a bit-level virtual model of the CAN capable of simulating the actual effects of background traffic and error occurrences.

(Note: we consider that the behaviour of the network in the presence of fault is simulated. However, in the particular case of CAN, analytical models of the CAN bus latency are already available (e.g., [10]), but simulation models allow to consider faults models of arbitrary complexity.

c. PWM motor control

Power is delivered to the rover motors via a H-bridge (4 transistors, see Figure 3) controlled by a PWM signal. The PWM duty cycle is computed by the motor regulator from the speed set point provided by the rover guidance controller and the actual speed measured using the wheels' optical encoders. The wheels' optical encoders generates quadrature signals acquired as frequencial input and then integrated for speed determination.

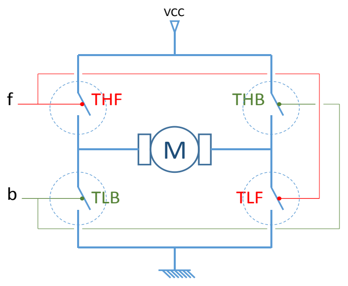


Figure 3: Motor H-Bridge control

The PWM controller was assessed under two perspectives: PWM timing and PWM design constraints both having effect on real time properties.

PWM control generation

The PWM generation device hold the following functional requirements (P4 - non exhaustive list)

- Frequency and resolution step of PWM control: 10kHz frequency with a 0.1 µs step)
- Latency for application of new PWM value (next cycle)
- Immediate desactivation of active state of the PWM (lower than 1 ms)

Compliance to the previous requirements may be achieved in many different ways, including:

- Pure software implementation toggling an output port (resolution will be certainly an issue)

- Pure hardware specific implementation with single register interface and fixed frequency (too specific solution)
- Mix mode with a simple timer, control by software interrupt and latch of an output (resource will be an issue)
- Hardware dominant solution with multiple register interface but "reasonable" or limited resource (most realistic in this basic example)

Attached to selected scenario we have also a list of design constrainst (C4) stemming from domain experience, available resources, etc. For instance :

- Register shall be 32-bit wide. At most 4 registers must be necessary (duty cycle value, frequency value, cycle counter and register control)
- The timer frequency range shall be in [100Hz,100KHz] (10kHz for motor control)
- The effect of the frequency and duty cycle change shall only occur at the next PWM cycle

For the evaluation of the motor controller we will consider the failure mode "wrong design" (F2M1).

The interaction between the PWM and the H-bridge (the device that physically controls the power signal) raises another possible failure mode. As the H-bridge provides no overcurrent protection, care shall be taken not to switch transistors located on the same side of the bridge on "at the same time" in order to avoid shortcuts. This situation can be prevented by introducing a *silence time* between the commutations of opposite transistors. The duration shall in particular take into account the transistor switching time. The respective failure mode are noted F1Mx.

The following property shall hold:

$$\forall\, tb_i, tf_j : \left|tb_i - tf_j\right| \geq \Delta t_{dz} \qquad \text{(P5)}$$

Where tb_i (resp. tf_i) be a time at which signal "backward" (resp "forward") is active, and Δt_{dz} is the duration of the "silent zone" where no transistor is active.

Failures modes

- F_{F1M1}: Erroneous Δt_{dz}, underestimation
- F_{F1M2}: Erroneous Δt_{dz}, overerestimation
- F_{F2M1} : Wrong design

Probability of occurrence

- O_{F1M1}: LOW
- O_{F1M2}: LOW
- O_{F2M1}: MEDIUM (component are selected to fulfill all forecoming applications)

Effects

- E_{F1M1}: During a rotation sense change, the opposite MOSFETs of the H-bridge transistors are on at "the same time". Cost impact is VERY HIGH: this configuration is basically a shortcut. Depending on the duration of the shortcut, the dissipated energy may leads to the destruction of the two transistors and the power supply. No other function is affected by the error.
- E_{F1M2}: During a rotation sense change, the PWM signal is delivered slightly later to the H-bridge. Cost impact is NEGLIGIBLE: the rotation of the wheel is slightly delayed which has no further impact on the rover's capabilities.
- E_{F2M1}: In case of wrong design with non capable component the function performance must be downgraded. In the worst case, a complete redesign may become necessary. Cost impact is HIGH because the problem will be detected during verification but it will have an impact on overall product planning.

In the absence of dedicated detection means, F_{F1M1} and F_{F1M2} are likely to be detected only in operations. F_{F2M1} will be detected lately during product verification meaning high effort for redesign.

Probability of detection

- D_{F1M1}: LOW. Depending on the duration of the "short-circuit", the number of forward/backward commutations, the error may stay undetected for a long time. However, it may eventually reduce drastically the lifetime of the transistor / power supply.
- D_{F1M2}: VERY LOW.
- D_{F2M1}: HIGH (as processus for verification are mature)

Cost of correction

- C_{F1M1}: LOW. The correction is basically a modification of a constant in the PWM management code.
- C_{F1M2}: LOW (same as C_{F1M1})
- C_{F2M1}: HIGH to VERY HIGH

For the "silence time underestimation" fault model, the combination of a VERY HIGH cost, a LOW detection probability and a LOW correction cost leads the introduction of a new detection means. This means shall have a MODERATE cost and HIGH detection coverage.

In practice, we created a virtual platform to host the PWM driver software and implemented a system-C observer to check (P5).The virtual platform shall provide a representation of time "compatible" with the property at stake. Here, we used a SystemC AT model of the PWM hardware driver.

For the "wrong design" fault model, the combination of HIGH cost, HIGH detection and HIGH (to VERY HIGH) correction cost leads to the introduction of a new detection means aimed at securing later development phases. As the functional requirement (P4) and design constrainst (C4) are expressed in terms of hardware and software interactions, the virtual platform shall at least provide a LT abstraction.

To facilitate the verification of the functional properties (P4) a functional model is developed (initiating also the test bench environment). From this level, requirements and associated properties are refined and allocated. During the decomposition process, some requirements may become “contracts” binding the different components of the motor control chain.

Figure 4 depicts some of these contracts:

- The PWM controller shall ensure a silence-period greater than Δt_{dz} *and* the H-bridge transistors shall switch is less than (e.g.,) $\Delta t_{dz}/100$ (P5).
- The power dissipated by the electrical motor shall be less than 15W. Therefore, the PWM duty cycle shall never be greater a given ratio for a given time. This contract propagated from motor power dissipation constrainst is not considered in the article.

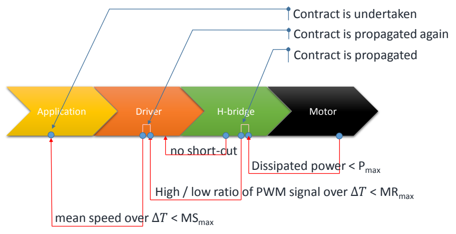

Contract is undertaken
Contract is propagated again
Contract is propagated
Application
Driver
H-bridge
Motor
no short-cut
Dissipated power < P_{max}
High / low ratio of PWM signal over ΔT < MR_{max}
mean speed over ΔT < MS_{max}

Figure 4: Motor drivers contracts

Another contract defined during the PWM controller design binds the hardware and software. It concerns the static and dynamic definition of the interface (see property C4). This contract is verified at the LT and AT modelling levels.

Section 5.b describes the use of the contract verification on the PWM controller.

d. The virtual platform

VLAB™[2] is used to develop the models and build virtual platforms. It provides a programmable and interactive environment for the assembly, configuration, programming and operation of electronic system level simulations, such as virtual system prototypes and virtual platforms, as well as other applications (see Figure 5) . A virtual platform integrates together simulation models and other simulation objects, scripts, tools, test and infrastructure software, and target software.

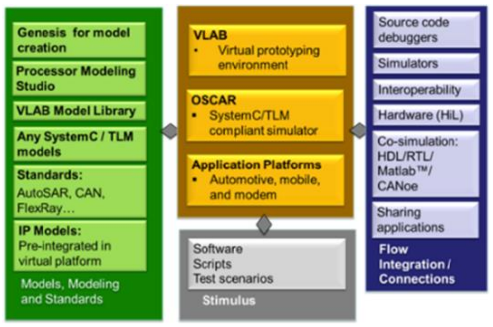

Genesis for model creation
Processor Modeling Studio
VLAB Model Library
Any SystemC / TLM models
Standards: AutoSAR, CAN, FlexRay…
IP Models: Pre-integrated in virtual platform
Models, Modeling and Standards
VLAB
• Virtual prototyping environment
OSCAR
• SystemC/TLM compliant simulator
Application Platforms
• Automotive, mobile, and modem
Software
Scripts
Test scenarios
Stimulus
Source code debuggers
Simulators
Interoperability
Hardware (HiL)
Co-simulation: HDL/RTL/ Matlab™/ CANoe
Sharing applications
Flow Integration / Connections

Figure 5: Tool organization

[2] VLAB^{TM} is a product of ASTC

The development of models does require a solid expertise understanding SystemC concept and language (object oriented). Moreover, the iterative refinements of models may be achieved by different persons with different coding skills.

The tool framework facilitates the creation of models thanks to a Genesis library compliant with SystemC and IP-XACT standards. It allows to capture model structure and connectivity and to automatically create C++ skeleton for the implementation of the behavioral part of the model. This library is also available in Python meaning models can be developed and debugged in a much faster and simpler way than in C++.

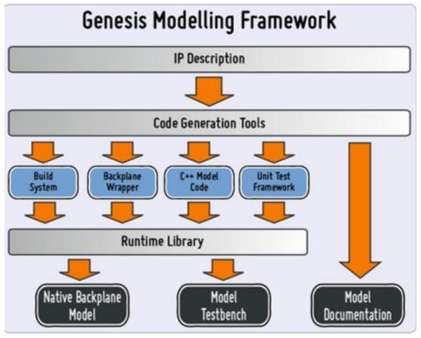


Figure 6: Genesis Framework

Finally, the tool provides an integrated and interactive execution platform leveraging again Python capabilities. It provides a simple yet efficient user experience for assembling virtual platform, debugging virtual platform (setting HW & SW breakpoints) and scripting test scenario (including fault injections in a non intruisive way) .

Modelling and assembling a virtual platform in Python are the key functionality used to support the implementation of the case study.

5. Implementation approach

a. Clock synchronization

We used ASTC VLAB$_{TM}$ “CAN toolbox” to create a model of our computation and communication infrastructure (5 nodes) and to inject faults during the execution of the clock synchronization protocol. A CAN node comes with two models, at token and bit levels. The toolbox provides an API for the configuration and control of nodes, including activation, frame transmission and frame reception. Python scripting allows to access directly to API, to send CAN frame and control the bit engine in order to introduce error in the CAN frame. These features have been used to exercice the clock synchronization protocol in situations where bit-level errors lead to multiple retransmissions of the same message. Such situations would have been difficult to obtain using the actual hardware.

b. The Motor control

The PWM controller scenario, as elaborated in section 4.c, is realized using VLAB^{TM}. Some parts of the complete model were developped using SystemC and Python modeling capabilities while some other parts were directly taken from the hardware library (for TLM transactor or MPC5674F AT models). The modelling scope for contract verification of the PWM controller is enlarged to the overall motor controller feature. For the sake of the demonstration, we consider that the C code of the motor controller was generated from the same Simulink® model used to design and validate the controller.

The development phase of the motor controller corresponds to the three predefined modelling style U, LT and AT.

Untimed model

The Untimed model is a functional model. It describes how the PWM signal is generated from the rotation speed setpoint and the actual speed measured from the optical encoders (see Figure 7 below). The Hardware Abstraction Layer API (HAL) is defined at this level. It allows to implement a static interface between application layer and driver abstraction using physical data interface with the motor regulation algorithm (% for PWM and speed in km/h for integration of quadrature signal).

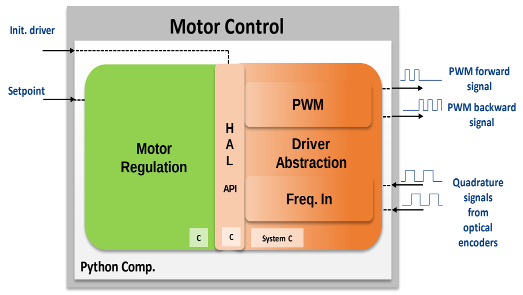


Figure 7: Functional model of Motor Control

The PWM controller is integrated in the driver abstraction SystemC module by implementing two specific C++ methods for the definition of the API (*init_mc_driver() and put_mc_pwm(int32 value)*).
The SystemC module integrates the driver abstraction with the PWM controller and the motor regulation. It declares two public methods *init_mc() and put_setpoint(value)* to allow access to PWM controller and to write into the setpoint data.
It includes one thread (or method) activated every 10 ms calling *c_mc_ctrl() for regulation* control activation. Using the toolset, the SystemC motor control model is first wrapped up into a python module which is then imported into the tool environment as a new component of the virtual platform. To use this newly defined component, the user first instantiates and initializes it. Then, he/she sets the setpoint value using to the python API *(driver_obj=vlab.get_instance("MC").obj, driver_obj.init_mcr(), driver_obj.put_setpoint(value))* .

This first level of modelling is a necessary step to build the subsequent models. We take benefit of it to perform a first verification of the design with respect to properties P4 and P5.

To do so, a test environment (or testbench) is built as shown on Figure 8. In particular, property P5 (dead zone) is checked using a dedicated SystemC/python component (P5 checker on the Figure 8) that is connected to PWM output signals via a monitor component). The test driver generates a scenario where wheels are moved forward and backward.

To close the control loop, a very simple model of the motor / encoder devices is developed and integrated in the platform. In this model, the frequency of the quadrature signals is considered directly proportional to the value of the PWM duty cycle (this is a good approximation as far as the frequency of the PWM signal is sufficiently high). This model is used to get a first insight on the performances of the regulation (response time, stability). In the future we will investigate co-simulation with Simulink® dynamic of motor itself.

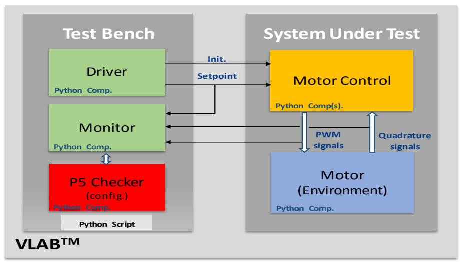


Figure 8: Test bench environment

To observe the results of the P5 verification, a trace is placed on the P5 checker diagnosis output port using tool tracing capabilities. See the trace of the scenario result in Figure 11.

Loosely Timed model

The Loosely Timed Model is a structural and behavioral refinement of the Functional Model.

It introduces several hardware and software components:

- The timer IP abstraction generates the clocks used to generate the PWM signal. This IP is modelled in systemC. It integrates all design constraint defined in C4 property (register size, frequency range).
- The driver abstraction provides the interface to the timer IP. This software component is defined in C. It is interfaced with the timer abstraction IP via a Hardware Software Interface (HSI). It integrates the C4 HSI contract (static and sequence interface).
- A SystemC bus driver interface allows to access to the timer IP peripheral. The C to SystemC interface complies with the TLM2 loosely timed standard. It is available in the tool component library.

This structural and behavioural refinement preserves the properties demonstrated on the functional model. It is used to express the Hardware Software Interface (HSI) contract (C4).

The HSI is implemented using five 32 bits registers:

- CNT running counter register (range of 24 bits) with 10ns resolution (range from 6Hz to 100 MHz, capable for 10KHz with 0.1% of precision)
- A and B compare registers for PWM control (range of 24 bits)
- CCR control register with enable bit (UCPREN bit 6), polarity bit (EDPOL bit 24) and fixed buffered mode (OPWFMB matching eMIOS definition bit 25-31)
- CSR status register with overrun bit (FLAG bit 31)
- Sequence transfer for buffered register A,B at the end of the PWM cycle and guarantee of B always greater than A.

The objective is not to implement the complexity of an hardware IP like the eMIOS, but build a simplified IP fulfilling the fonctionnality with respect of a reduced HSI accessed via TLM2 access (untimed).

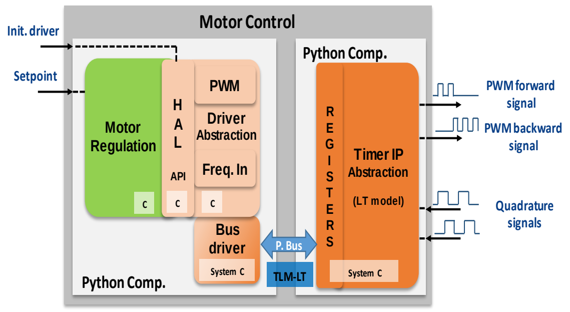


Figure 9: LT model of the Motor Control

The motor control is now split in two python components as depicted in Figure 9. This split allows the observability using VCD. The python components are then instanciated with the test bench architecture as in Figure 8. For the testbench, the interface for the motor control is identical as the functional model. The P5 checker can also be reused to demonstrate the dead zone contract. See Figure 11 for trace of the results.

The HSI interface contract can be demonstrated thanks to respect of the API resolved during virtual platform building and sequence demonstrated by instrumentation capability to trace TLM transaction (register access and value transported).

Approximated Timed model

The last model integrates approximated timing on bus communication and on hardware IP resource component access (internal definition and average processing time in hardware IP). Model can also complete the register interface when extra control registers are necessary for pre-implementation constraint, such as merging several LT models into a more complex and configurable hardware component. In any case the predefined HSI contracts shall be preserved: at least the same registers are used to interact with the component, according to the same sequence (protocol).

A use case more relevant than timer IP could be an image processing hardware accelerator merging LT models with modeling of internal timing interconnection.

In our experiment, the LT timer IP is mapped to the MPC5674F eMIOS AT model. The Timer IP is connected to a bridge and to the memory bus of the PowerPC core via the peripheral bus (see Figure 10). The target software is the binary software, including final drivers, application software compiled for the µC target.

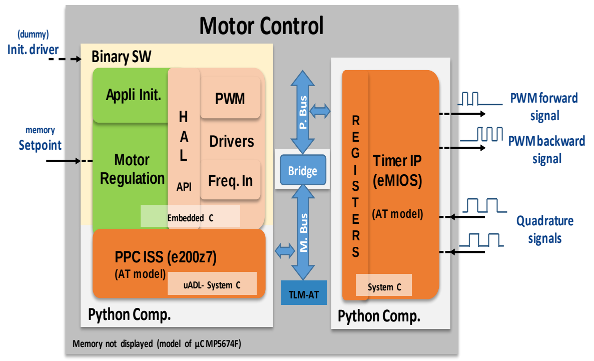


Figure 10: AT Model of the Motor Control

The test bench is only slightly adapted with respect to the one used at the more abstract levels. A python transactor component is inserted and connected to the same test bench driver. It integrates one method for target memory introspection using debugging API feature of the tool environment (*write_memory()*) and a second one to dummy the motor control initialization as it is now performed by the µC start-up code. The simulation scenario generator and the property checker component do not change. The P5 checker can be reused for the validation of the contract defined in the FMECA process. The functional property P4 of the PWM control is verified. The C4 contract on the HSI register access and sequence is also verified.

The simulation trace typical of every refinement level is depicted below in the Figure 11.

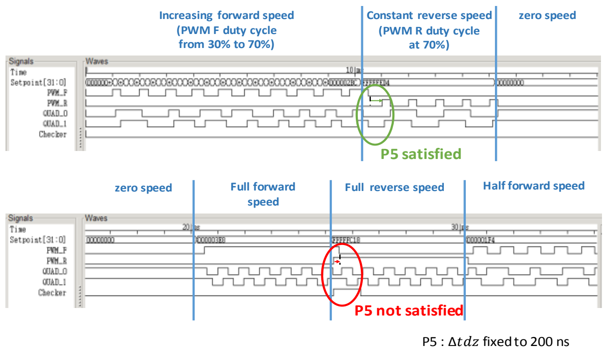


Figure 11: Trace of dead zone validation

6. Conclusions and future work

In current SystemC ecosystem, the adequacy of model representativity for verification objective and the quality of the simulation models curbs the large deployement of virtual platform. Our approach provides a first level of guidance and formalization of the modeling process.

The “FMECA-like” approach introduces – to some limited extend, however – objective criteria to determine whether a refined model is necessary or not. It allows to start the modelling phase at the adequate abstraction level for the verification of the selected property. The formalization of the development and the test of an architecture compliant with SystemC modelling standard allows a sound design covering hardware software codesign process. We demonstrated that test elements can be reused all over the model refinement. Moreover, the contract approach provides backbone for decomposition and guarantee on the coverage of the requirements.

The tool automation features simplify the platform models generation for component interfaces description and component bindings. However, for the creation of a new component model, the remaining manual operations (code implementation, platform integration and test campaign) still represent around 90% of the development costs. In actual practice the elaboration of simulation models follows a typical top-down approach based on successive abstractions. Such refinement has been formalized through the control motor modelling experiment. More important, the models have been continuously verified. In our case, the development and testing of two additional abstracted models is only estimated at 10 to 20% of the overall effort.

Balanced to the model development effort, the proposed methodology with virtual platform provides strong benefits on the overall product development. The contract and the model based communication facilitate inter domain communication and improve the design quality of the product. It helps to reduce significantly the re-work iterations. The effort gained can be estimated at 10 to 20%. This was demonstrated in our experimentation and also on porting the OS Trampoline [15] on MPC5674F microncontroller for twIRtee. The early hardware software integration by simulation reduces operational set up and validation.

The approach proposed in this paper will be experimented for the development of other parts of the INGEQUIP project demonstrator. In particular, the camera-based line tracking function will be developed according to the same approach. This function requires a lot of computing power, focus will be placed on the issue of hardware/software design space exploration.

7. Acknowledgement

The authors thank all people and industrial partners involved in the Ingequip project. This work is supported by the French Research Agency (ANR) and by the industrial partners of IRT Saint-Exupéry Scientific Cooperation Foundation (FCS): Actia Automotive, Airbus, Airbus Defence and Space, ASTC Design Partners, Continental Automotive, SAGEM, Systerel, Space Codesign Systems and Thales Avionics.

8. References

[1] SystemC core langage definition, Accelera Systems initiative standards, http://accellera.org/downloads/standards/systemc

[2] B. Koppelmann and all, “An open and Fast Virtual Platform for Tricore^{TM} based SoC Using QEMU, DVCON Europe 2014

[3] ESA SockROCKET virtual platform, http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SoCROCKET_Virtual_Platform_-_SystemC

[4] Freescale ADL, uADL open source project, http://opensource.freescale.com/fsl-oss-projects/

[5] Franck Schirrmeister, Synopsys, “System Level Market trends”, 2010 Synopsys Interoperability forum

[6] P. Cuenot, N. Tavernier, JM Talbot, “Embedded Software V&V using virtual platforms for Powertrain application”, ERTS 2008 Toulouse, France

[7] V. Lefftz, J. Bertrand, H. Casse, C. Clienti, P. Coussy, L. Maillet-Contoz, P. Mercier, P. Moreau, L. Pierre, et E. Vaumorin, « A design flow for critical embedded systems », in 2010 International Symposium on Industrial Embedded Systems (SIES), 2010, p. 229–233.

[8] F. Herrera, H. Posadas, P. Penil, E ; Villar, F. Ferreo, R. Valencia, G. Palemro « The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems”, Journal of System Architecture, volume 60, issue 1, Jan 2014.

[9] L. Rodrigues, M. Guimarães, and J. Rufino, ‘Fault-tolerant clock synchronization in CAN’, in Proceedings of the 19th Real-Time Systems Symposium, Madrid, Spain, 1998, pp. 420–429.

[10] I. Broster and A. Burns, ‘Timely use of the CAN protocol in critical hard real-time systems with faults’, in Real-Time Systems, 13th Euromicro Conference on, 2001., 2001, pp. 95–102.

[11] Electronic Reliability Design Handbook, US Department of Defense, MIL-HDBK-338B, Oct. 1998.

[12] Road Vehicles – Controller area network (CAN) – Part 1: data link layer and physical signalling’, ISO/TC 22/SC 3N, 7-May-2014

[13] http://cache.freescale.com/files/32bit/doc/ref_manual/MPC5674FRM.pdf

[14] VLAB™, ASTC, http://www.vlabworks.com/

[15] Trampoline OpenSource RTOS projet, http://trampoline.rts-software.org/

QBox: an industrial solution for virtual platform simulation using QEMU and SystemC TLM-2.0

Guillaume Delbergue
GreenSocs -
Bordeaux INP, CNRS IMS, UMR 5218
guillaume.delbergue@greensocs.com

Mark Burton and Frederic Konrad
GreenSocs
mark.burton@greensocs.com
fred.konrad@greensocs.com

Bertrand Le Gal and Christophe Jego
Bordeaux INP, CNRS IMS, UMR 5218
bertrand.legal@ims-bordeaux.fr
christophe.jego@ims-bordeaux.fr

Abstract

As industrial demands grows for modeling, simulation and exploration tools during SoC design, there is a need for simulated CPU models that work with the other tools in the space design. There have been many attempts to build libraries of CPU design models, most notably the Open Virtual Platform project [1]. This work focuses on defining a complete open source SystemC compliant approach with a QEMU based model, which supports almost all current and past CPUs. QEMU [2] has many advantages not least a very large and dynamic open source community with hundreds of developers active all over the world. Our work makes QEMU available in a standard SystemC [3] (IEEE 1666) based tool environment. This article addresses the current state and upcoming features of two different integration techniques that allow the QEMU virtualizer and emulator to be used in a SystemC simulation context : QEMU-SC [4] and QBox (QEMU in a Box). This article also examines how these current implementation work. The limitations they have with respect to SystemC are also given. It will go on to look at recent developments both within QEMU itself and in the integration between QEMU and a SystemC that are aimed at improving simulation performance and usability of these solutions.

I. Introduction

The current embedded market is increasingly taking advantage of standard *Components Off The Shelf* (COTS), while adding value in software and complex Systems on Chip (SoC) IP. Within that context, driven by cost reductions and time to market, virtual platforms are essentials (to enable software teams) to become productive earlier in the design cycle. It also provides a common link environment between hardware, integration, verification and software development. In essence, virtual platforms reproduce system behavior, execution of target software, debug and development in the absence of "real" hardware platform. The virtual platforms can and should be used as a means for exploration between hardware and software engineers during development cycles.

The SystemC language allows hardware descriptions to be constructed in a *C++* based language. However, as the complexity of the IPs increases, the SystemC simulation environment is not necessarily suitable to provide suitably fast models. It is theoretically possible to simulate complex IP's such as CPU's within SystemC simulation kernel. But as we can see in SoCLib[5], performance can be an issue, especially when the processor is modelled at RTL level that is computationally intensive. A better solution for complex IPs like CPUs is to model it in a virtualizer or emulator and then to integrate the model into a SystemC simulation environment. Moreover, the TLM-2.0 (Transaction-Level Modeling) standard, which is an extension of SystemC, improves interoperability between memory mapped bus models. It also includes the notion of time quantum which was explicitly intended to assist with this sort of integration.

One such external virtualizer is named QEMU (Quick EMUlator). It is a generic and open source machine emulator and virtualizer which allows engineers and developers to execute their software binaries (operating systems and applications) made for one machine (processor and its peripherals) on another one. For example, the execution of ARM or PowerPC binaries on a x86 processor based computer. QEMU enables simulation of multiple kinds of processor cores and is based on a JIT (Just In Time compiler) code generation technology. JIT technology enables code recompilation / translation at run time. It can achieve emulation of the simulated processor core at near real time speed.

To interface QEMU (the CPU virtualizer or emulator) with a SystemC simulation environment containing SystemC models, multiple solutions have been proposed such as QEMU-SC, an open source solution. QEMU-SC aims to embed SystemC models in QEMU through a wrapper (as detailed in Section II-A). However, this purposed solution has drawbacks. The authors in [6] implement a solution to interface QEMU and SystemC simulation environments by separating SystemC and QEMU on two threads. The communication is made through a shared memory and a FIFO mechanism. However, the document doesn't detail time synchronization aspects between QEMU and SystemC simulation kernel based time. The solution in [7] integrates SystemC with both QEMU and OVP using a SystemC bridge. The QEMU/OVP simulations run on a own thread and the bridge is a set of C functions included in a library which is statically linked to the SystemC runtime library. The time synchronization between QEMU/OVP and SystemC is performed on a call to a SystemC model. It enables the update of SystemC time to be synchronized with the QEMU time. This solution breaks dynamic quantum (See Section III-A). A similar implementation [8]

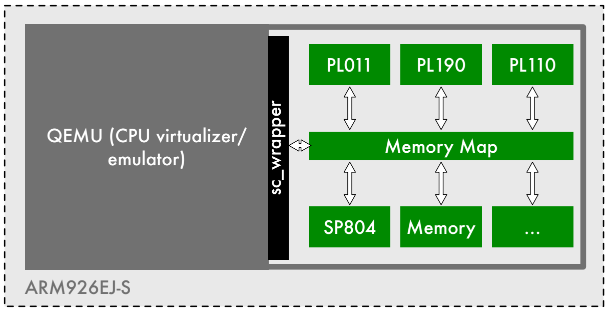


Fig. 1: ARM926EJ-S architecture with QEMU-SC

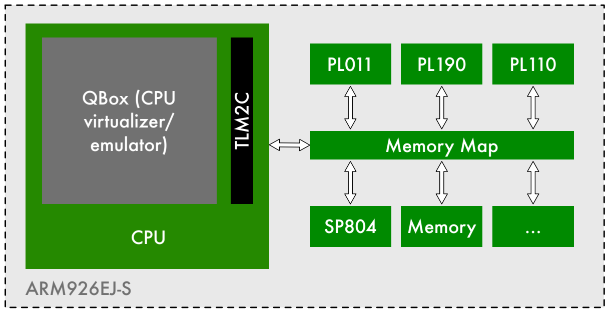


Fig. 2: ARM926EJ-S architecture with QBox

to QEMU-SC adds the capability to estimate the performance of a target system from system perspective. It is also able to trace all the information that are necessary for the estimation. In [9], BSD sockets enable communication between two virtual CPUs (two instances of QEMU) and SystemC models. A thread controller manages transactions from different cores to SystemC models. To synchronize the time, the authors send the simulation time in their transactions. However, there are no details about the algorithm used to synchronize all bases of time.

In all cases, SystemC is used to simulate complex SoCs including digital peripherals and one or multiple processor cores. The rest of the article is organized as follows. Section II briefly introduces and compares QEMU-SC and QBox solutions by presenting typical use cases. Then, detailed features, advantages and limitations of QBox for industrial virtual platforms are discussed. Finally, different solutions have been benchmarked and compared from virtual platforms to real hardware.

II. QEMU-SC SIMULATOR AND QBOX CPU VIRTUALIZER/EMULATOR

A. *QEMU-SC simulator*

Combining QEMU's ability to efficiency simulate processors with the flexibility of SystemC meets the needs of much of the embedded software industry, to provide a model environment based on standards with high performance. This environment captures the power of QEMUs CPU simulation environment along with the standard approach for writing models. QEMU-SC [10], based on works of the authors in [11], provides a solution to connect SystemC simulation environment with QEMU. QEMU-SC is an open source hardware and software emulation solution for SoC development, specifically targeted development of add-on cards or peripherals that have to be connected to an existing (off the shelf) platform.

In this context, QEMU is the "master" during simulation execution. It contains the off the shelf platform. It instantiates and controls the SystemC part. This approach enables models written in SystemC to be used from within QEMU. Models can be integrated using specific memory mapped addresses such that QEMU communicates through MMIO (Memory-Mapped I/O) or PCI interfaces. In this context, SystemC is used as peripheral models to a QEMU based software platform.

As show in Figure 1, QEMU-SC provides a QEMU hardware model named `sc_wrapper` to communicate between QEMU and a SystemC model. This wrapper handles reads and writes to the memory map by building TLM transactions and redirecting them to the SystemC side. Transactions are cached, but nonetheless, this incurs a pointer and several integer copies. While in general performance is an issue for TLM systems, this overhead is minimal as these devices are not typically accessed frequently. QEMU-SC also provides a wrapper for interrupts in both directions. `sc_platform` registers SystemC models within QEMU and binds them to TLM router and IRQ vector. Depending on the simulation requirements, two different implementations are available:

- MMIO which is a generic implementation in TLM-2.0 standard,
- PCI which is a PCIExpress bus implementation in TLM-2.0 standard.

The current implementation of QEMU-SC has some limitations in that IRQs are not handled as cleanly as they could be. It means that SystemC models need to have a 'handle' to the QEMU model, and pass that back to QEMU when they are sending an IRQ. This breaks the TLM standard and means that some alterations are required to a SystemC model to be used with QEMU-SC.

B. *QBox CPU virtualizer/emulator*

QBox is an integration of QEMU virtualizer and emulator in a SystemC model. Contrary to the QEMU-SC solution, QBox or QEMU in a (SystemC) Box, treats QEMU as a standard SystemC module within a larger SystemC simulation context. SystemC simulation kernel remains the "master" of the simulation, while QEMU has to fulfill the SystemC API requirements. This solution is an open source QEMU implementation wrapped in a set of SystemC TLM-2.0 interfaces. QBox allows the powerful JIT based

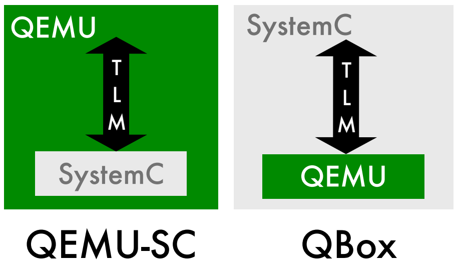


Fig. 3: QEMU-SC (QEMU is master, SystemC simulation kernel is slave) vs QBox (SystemC simulation kernel is master, QEMU is slave)

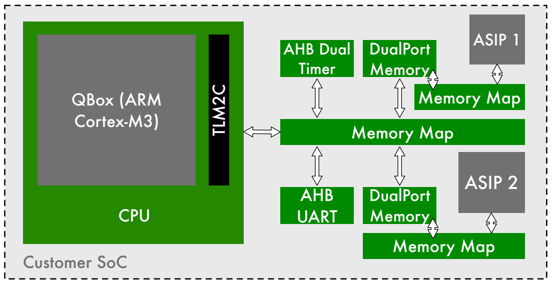


Fig. 4: Example of a virtual triple-core platform

CPU simulations to be totally exploited within a TLM-2.0 context. QBox is provided as shared libraries that contain QEMU based CPUs as shown in Figure 2. Each QBox library contains a specific CPU core and exports a set of TLM-2.0 like 'C' interfaces. A QBox library is instanced in a SystemC simulation context through the SystemC wrapper called TLM2C. TLM2C library provides the C++ TLM-2.0 standard interfaces. It exports TLM-2.0 'C' like interface to the standard TLM-2.0 C++ interface.

This solution integrates well into a virtual platform to facilitate the co-design of hardware models but also to validate the software which runs onto the platform. QBox is sufficiently flexible to support some basic system components from the QEMU library that have been included within the QBox component. So, only the models of interest can be specified in SystemC language. It means that QBox is a flexible platform simulation environment. Thus, novel components can be added and device drivers can be designed. QBox can be used within the context of a much wider system as mealy one of many components.

Figure 2 provides an overview of an existing virtual platform using the SoC ARM926EJ-S. It is the SoC used in the ARM Versatile PB board. Depending of the host machine, QBox emulates or virtualizes the core part of the SoC (an ARM9E-S) and connects its AMBA bus as a SystemC TLM socket, thanks to the TLM-2.0 standard generic protocol. As QEMU is written in C (as opposed to SystemC which is standard C++ class library), a wrapper called TLM2C is required to connect them. Each of the remaining Versatile PB models are modelled in SystemC language and connected to the bus using TLM-2.0 standard. To specify the bus routing, GreenRouter has been used. It is a part of the open source library GreenLib [12]. GreenRouter makes use of optional extensions in the TLM-2.0 protocol that are provided by GreenSocket. It allows routable addresses to be allocated in devices, rather than in the router. This is merely a convenience and ease the system description.

The platform described in the Figure 2 boots with a Linux kernel. It is possible to interact with Linux through the UART (PL011 model) and also view the output of the graphical interface PL110, which is part of the SoC. All parts of the Figure 2 are available and open source[13].

C. *Use cases*

QBox and QEMU-SC both have their uses. QEMU-SC is useful in cases where cards or a small number of components are under development. It is also useful when components have to be added to existing off the shelf platforms. QBox is more suitable when the CPU element is considered to be one component of a wider system. The main difference between QBox and QEMU-SC is that:

- For QBox CPU virtualizer/emulator, SystemC simulation kernel is the "master" of the simulation, ie QBox, which embeds QEMU, acts as a slave, a SystemC model. SystemC simulation kernel controls the QEMU execution. SystemC models can be interfaced without modifications.
- For QEMU-SC simulator, QEMU is the "master", ie QEMU pause and resume SystemC simulation kernel if necessary, SystemC models are slave. SystemC models have to be adapted to the QEMU interface.

The Figure 3 illustrates differences.

Critically, both QBox and QEMU-SC solutions enable execution of software binaries (operating systems and application stacks) without any modification to real target code. In both cases, QEMU directly executes the code from memory. For QEMU-SC, the memory is held within QEMU. For QBox CPU virtualizer/emulator, the memory is held within SystemC and accessible via a TLM-2.0 interface. QBox makes use of the TLM-2.0 Direct Memory Interface (DMI) to access memory. Thus, the execution speed of a binary in QBox, which doesn't use excessive IO (as an access to a register through the memory map), is typically indistinguishable from native QEMU. QBox's performance is typically extremely fast:

- for non natives guests (e.g. ARM on x86 processor), it can be within an order of the host machine. It is typically "real time" for most embedded processors,
- for native guests (e.g. x86 on x86 processor) the KVM (Kernel Virtual Machine) can be used. This provides simulation speed close to host performance.

However, there exists run-time penalties when the software running in QEMU accesses IO. Currently, the time to perform an IO as a simple write to a register from Linux is about 40ms (host time) on an host machine with an Intel Xeon E3-1271. For example, Linux kernel executed in QEMU performs a write to a register modelled in a SystemC model like one of the registers in PL011 peripheral). Looping on IO constantly suffers from these penalties. However, this is typically not the behaviour of most systems. Actually, most systems do exhibit this behaviour, typically they are spinning waiting on IO, so the delay isn't important either. Nonetheless, this is an area where we intend to do further research. We expect to decrease this number to few nanoseconds if the IO is thread safe. This issue is addressed in more details below (See Section III-B).

QBox is designed to fulfill today's requirements for virtual platforms. As QBox provides a TLM-2.0 component instance, it is ideally suited to support the increasing demand for platforms that integrate multiple homogeneous and heterogeneous processors. As QBox is a TLM-2.0 component, it is clearly possible to mix heterogeneous CPUs within a platform to achieve, for example, Asymetric Multi-Processing (AMP). This is a feature that is missing in QEMU (and QEMU-SC). These sorts of AMP systems are extremely typical in everything such as smartphones, Internet of Things (IOT) or smart devices. A typical system would contain an 'off the shelf' virtualized with QBox and one or more ASIPs. All of this executes within SystemC simulation. However QEMU does support homogeneous cores working in a Symmetric Multiprocessing (SMP) manner. This is also possible within QBox. Hence both homogeneous (SMP) and heterogeneous (AMP) systems can be modeled and simulated by QBox.

Currently, QBox has one limitation for multi cores support. All cores of an SMP CPU need to run in the same instance of QBox. QBox doesn't export into SystemC the inter-CPU communication system necessary to enable SMP to be simulated within SystemC. This would be an interesting arraignment if more complex interconnect structures have to be investigated. For instance, when some of the CPUs have access to certain peripherals. This is probably a minority of use cases right now. But nonetheless we would like to enable it.

1) Example with a triple-core platform: QBox has been used with success in the definition of a virtual platform containing an ARM Cortex-M3 with two ASIP models for a future IOT chip. This virtual platform is composed of standard Cortex-M devices modelled with SystemC like AHB UART but also some custom IP models as described in Figure 4. The communication channel between the two subsystems (ASIP and Cortex-M3) in this design is typical of many such designs revolving around the use of interrupts and shared memory and this aspect has been modelled. However, care must be taken in a quantum based system that both sides are reactive enough to those interrupts to enable the smoothly communication. As it is typically the case with quantum based TLM-2.0 models, some degrees of quantum 'tuning' are required to balance performance and communication accuracy.

This virtual platform use multiple threads to speed up simulation execution. It makes a better usage of resources available on the simulation host. The ARM Cortex-M3 is embedded in a QBox instance. QBox runs in 2 threads (one for IO and one for CPU, as QEMU). Another thread is used by SystemC itself. We recall that SystemC is the master of the simulation. As the ASIP model is provided by a third party it is outside our control.

This virtual platform has been developed in parallel with the real SoC. The platform has already shown benefit in terms of hardware and software teams that agree on their (complex) interface between the ARM and ASIP systems. They are now able to develop their softwares ahead of the hardware tape-out. This is a classic example of the Electronic System level (ESL) working, with a direct return on investment for a small device, based exclusively on open source infrastructure.

It should also be noted that, there is no agreement on how signals outside the memory mapped bus are handled. The third party devices in this case used a typical implementation of interrupt signals. Unfortunately, it does not play especially nicely in a quantum based TLM-2.0 simulation context. Of course, this only required wrapping of those interfaces. But nonetheless this is annoyance and confusion that nobody needs. We hope to address this issue in a future work.

III. Features of QBox

A. Time synchronization

QBox works within a SystemC simulation. QEMU's notion of time is typically based on the guest clock, also called `vm_clock` in QEMU. On the other hand, SystemC is purely event driven and its clock moves as fast as events can be processed. Due to different time domains, it is necessary to ensure time synchronization. To further complicate matters, QBox and SystemC run in a separate threads to improve efficiency and simulation speed. QBox takes advantage of the quantum mechanism built into TLM-2.0. This enables a model to be at most one quantum ahead of SystemC's current time. QBox's time increases in parallel to SystemC simulation time in a different host thread. Quantum level synchronization is maintained between threads by ensuring that Equation (1) is always respected.

$$|time(QBox) - time(SystemC)| \leq Quantum \tag{1}$$

Time synchronization is one of the bottlenecks of a virtual platform. Indeed, due to the static length of a quantum, it is necessary to add checkpoint quantums even if there are no events pending in SystemC. In some systems, to handle tightly coupled IO, the quantum has to be reduced. But, this has a negative impact on performance because it increases simulation time. It can also be wasteful if IO is not always used. To ensure this, when QBox or SystemC executions are at a quantum boundary, they use a thread *wait* to synchronize. It enables to wait for the other parties to finish their quantum.

Currently, QBox only supports static quantums. However, in order to avoid redundant and unnecessary checkpoints, we speculate that a dynamic quantum mechanism may be beneficial. At this point, this is left as a future work. A straight forward dynamic quantum approach, in which a 'synchronization' - or quantum - would be placed into both SystemC and QEMU. At each point, their potential interaction (but no un-necessary points) would likely have a positive effect on a single CPU system. Our concern is that on a multi-core system, IO events on one core would cause un-necessary synchronization points on other cores. Therefore, our conclusion is that a 'simplistic' approach is not going to have universally positive results. Therefore we conclude that this is a subject for much more detailed research.

B. *Performing IO*

QBox accesses memory by using standard TLM-2.0 transactions. When the guest software running in QBox wants to perform an IO to a SystemC model, it stops its execution. Then, one of the following two cases will occur:

- If SystemC is running, a thread safe event is posted to the SystemC simulation kernel. Then the simulation scheduler has to run the TLM-2.0 IO request
- If SystemC is sleeping (as it has already finished its quantum), SystemC will be woken up. The same thread safe event is posted.

Once the processing of the SystemC event is completed, QBox is notified. Then it continues its execution with the completed transaction. The Figure 5 illustrates both cases. Transactions from SystemC to QBox are used for interruptions generated from devices (SystemC models). QEMU is able to receive these interrupts in a thread safe way. Hence, no special care has to be taken when they are generated by devices and sent by using standard TLM-2.0 based sockets.

When QBox wants to access an IO managed by a SystemC model, it does so by issuing a thread safe event which is executed by the SystemC simulator, within the SystemC thread. This guarantees that only the SystemC thread is used to execute SystemC code. This solution also maintains the single threaded nature of SystemC. However, this adds complexity and decreases performance of simulation when different simulators have to be synchronized. It may be possible to use some of the constructs in work on parallel SystemC [14]. But this falls outside of the existing SystemC standard.

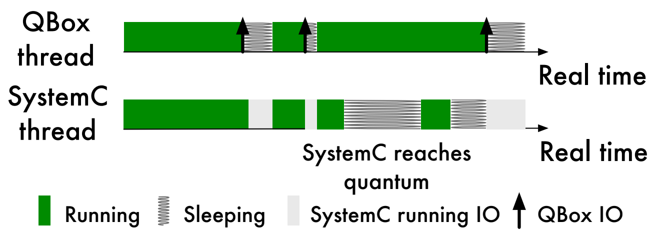


Fig. 5: QBox synchronization illustration

In general, it is possible to write thread safe SystemC models. The models own data structures that can be protected in the normal way. As the SystemC language is not guaranteed to be thread safe, models that require accesses to the kernel need to use an asynchronous notification. If we could guarantee that the models connected to QBox were thread safe, it is not necessary to switch threads. In this case, QBox would be able to directly access IO without the synchronization mechanisms. To add this feature we would like to add a new way to inform QBox that this part of SystemC simulation is thread safe. It means that it can be directly accessed without interruption. We do this while complying with the TLM-2.0 standard. This will speed up simulation and improve usage of parallel threads.

C. *From mono-thread to multithread*

Currently, QBox (and QEMU) run within two threads : one exclusively for IO operations and another one for CPU execution. The CPU thread runs the TCG (Tiny Code Generator) which performs the translation from CPU guest code to CPU host code. This is the JIT engine (or dynamic binary translator), which is the core of QEMU. Initially, QEMU was proposed to emulate one CPU on a host with a single CPU. When multiple CPUs are emulated, a "round robin" approach is used within the single CPU thread. However, with the proliferation of cores in both the PC hosts and the platforms being modelled, this approach is not optimal.

Previous works like [15] forked QEMU to run each virtual CPU on its own thread. To do that, it was necessary to parallelize the TCG [16] part of QEMU. The TCG performs two steps. On one hand, it transforms any supported guest CPU target instructions to TCG operations. On the other hand, it transforms TCG operations into CPU host instructions. This intermediate step between guest and host translation decreases CPU dependencies by adding a new level of indirection. It makes the TCG flexible and maintainable across a large number of hosts and guests. Unfortunately, parallelization is difficult to implement. The authors limited themselves to only one small case. Therefore, this approach is not safe for all cases.

In all cases, there are a number of issues that have to be addressed. One of the most obvious is the issue of 'atomic instructions'. For a single threaded model, the model can assume an ordered memory model. It can implement atomic (compare and exchange, load/store exclusive etc - used to synchronize multiple cores) with no special restrictions. However, this is not the case for multi-threaded models.

The authors of COREMU[17] emulate multiple cores by creating multiple instances of existing sequential emulators. A thin library layer is used to handle the inter-core, the device communication and the synchronization. The objective is to maintain a consistent view of system resources. However, they did no more than adding a mutex around the load and store exclusive instructions for ARM. Unfortunately, this is not sufficient. Indeed, it does not fulfill the ARM requirements, and fails to support anything but the smallest of guest code sequences. The authors of HQEMU[17] combines LLVM with TCG to speed up translation. This solution not only speed up single threaded applications that runs in the guest but also multi-threaded applications.

D. MTTCG project

The MTTCG project (a QEMU project) aims to allocate one host thread to each simulated CPU to significantly improve performance. It also enables the power of the host machine within the heart of QEMU itself. This project also aims to upstream changes so that the techniques are re-donated to the community such that others can build upon it. For this first iteration, MTTCG focused on the ARM architecture. But it can also be extended to all targets within QEMU. The initial limitation is only due to a single patch serie that addresses atomic instructions within the ARM architecture. Other architectures are being addressed in parallel.

1) Global TCG State: In the current implementation of QEMU, there is no protection against two threads attempting to generate code at the same time, placing the results into the translation buffer (a part of TCG). In a multi-threaded system, it leads to corrupted code generation from time to time. In order to better handle the translation cache, the key question is whether the translated code cache should be per-guest-CPU (per-thread) or global (shared between multiple guest CPUs). Per-guest-CPU means less possibility of locking contention. But it means more overhead generating code. Every time the guest reschedules a process to another guest CPU, we'll have to translate all its code all over again for the new CPU. A strictly global cache is not a great idea either. Indeed, it won't work if we eventually moved to supporting heterogeneous systems (for example, one ARM CPU and one SH4). One possibility is a hybrid system. Each guest CPU have a pointer to its TCG cache (which could then be shared between several other CPUs).

Sharing a cache would allow us to take advantage of code that is translated by one core and then used by another. On the other hand with one cache per core, updates on the caches with a lot less locking can be performed. Each CPU could generate translations simultaneously for its individual cache. However, invalidates can be done across all the caches if any core writes to program memory. This would be expensive as all CPU caches would have to evict TBs, rather than a single 'central' entry being evicted.

Interestingly, the structures existing in QEMU are similar to a tiered cache system in hardware. While each CPU holds a local list of pointers to translated code, there is also a wider 'level 2 cache' shared by all CPUs. The local 'caches' are simple indirections to the main cache, so 'evictions' can happen centrally. This provides the benefits of tiered caching. It permits to implement a highly optimal solution. In its current form, only one CPU can generate a TB entry. Then, it is posted both locally and to the global TCG cache. This already provides a highly powerful solution. It would be possible to enable more than one CPU to generate different TB entries concurrently.

2) Dirty tracking: Currently, QEMU handles guest writes to memory by using a set of bitmaps for tracking dirty memory of various kinds setting the internal QEMU TLB entries up to force subsequent reads (e.g. for code execution) to consult the real memory (this is termed the slow-path).

This is a fairly long sequence of operations (guest write; read bitmaps; update TB cache structures; update bitmaps) which is currently effectively atomic because of the single thread being used. In order to enable multiple threads, this chain has to be dealt with carefully. Specifically, when a CPU marks an area as 'dirty' in the bitmaps, all CPU's need to see this message. QEMU works by assuring that the effect of the JIT TB cache can not be seen. Therefore, dirty information has to be transmitted and acted upon 'immediately'. In order to achieve this, a new service was necessary to be added to QEMU to allow a core to request that all CPU's stop. For instance, a TB entry was flushed atomically from the point of view of all simulated cores. This new mechanism is introduced in our work.

3) Memory consistency: Host and guests might implement different memory consistency models. For instance, the ARM memory model does not guarantee that other cores will necessarily see writes in order (or at all), until a memory barrier is executed. An x86 based machine has a stronger memory model. It normally guarantee that all threads see all writes. While supporting a weak ordering model (eg. ARM) on a strong ordering backend (e.g. x86) isn't a problem. For now, the only realistic solution is to place memory barriers after writes. The performance impact of this very much varies in terms of exact architecture and implementation. We do not present benchmarks here. Indeed, they are being worked on by other members of the QEMU community. As our work initially focused on ARM running on x86, we were able to ignore this issue for the moment.

4) Instruction atomicity: Atomic instructions allow guest code to run on several guest CPU's for the synchronization. They are the critical means by which e.g. SMP is achieved. They rely on a few basic instructions being 'atomic' from the perspective of all guest cores. One example is a store exclusive instruction in ARM where the store should succeed if no other thread is active (either read or written) to a memory location since a preceding load exclusive operation. One obvious implementation of these instructions is to check that the value remains the same between the load and store exclusive operations. Unfortunately, it is not architecturally correct, since the architecture specifies that no load or store should have occurred. Potentially a non exclusive load or store could have occurred. It may not effect the value in the memory. A subsequent store exclusive would then erroneously succeed.

Experimentally, we have replaced the store exclusive code that was written in this manner by host atomic instructions (e.g. using the `cmpxchg` primitive). We have shown it seems to be stable across many boots and application cycles. However, theoretically even using host atomic instructions is not necessarily architecturally correct in all cases. A more 'complete' solution would be to mark memory locations in the dirty bitmaps and force access to them to be taken through the safe-work mechanism described above. While this is conceptually much cleaner, it may have performance impacts on simulation speed. At this time, other ones in the QEMU community are building such an implementation based on our mechanisms to test out the potential speed reduction impact. Thus, host atomic instructions has not caused any issues in terms of atomic locks not working correctly. Test cases to show how locks could be broken have been attempted, but we have yet to produce the effect. Nonetheless, it makes sense to model the architecture more accurately if the performance penalty is acceptable.

E. Scientific contributions

Our approach is to provide usable virtual platforms for multi-core systems. Such systems have to to be simulated across a number of host threads. We remain at the "CPU block" level by mapping virtual cores onto physical host cores. This is the major difference with other approaches that focus on SystemC threads. It is driven by synchronizing the number of CPU cores.

During the investigations, we have identified and provided solutions for 2 JIT engines. Specifically a new approach proposed coordinate cache-coherency work within a JIT, between cores. We term this the safe-work mechanism. It allows cores to request work to be done locally, on specific remote cores, or globally. The mechanism itself is robust. We have identified when and how it should be deployed within the JIT for specific cache operations. One good finding is that the safe work mechanism, and the points during simulation when it should be deployed can be implemented in a architectural neutral way.

To solve the issue of the shared instruction cache, we discovered that the way in which the JIT cache structure operates actually mimics the effects of a real hardware cache. Because of this, we strongly favoured the approach of sharing a sort of level 2 cache between CPUs (much as is common in real hardware). Implementing this yields extremely impressive numbers (see Section IV).

Finally for atomic instruction handling, we re-used the hosts ability to perform atomic instructions, effectively translating guest instructions directly to host atomics. In our case, we restricted ourselves to the compare and swap (`cmpxchg`) atomics. This is clearly guest dependent. For our case, we focused on ARM on X86. However, we concluded that this solution has the drawback that some guest instruction semantics are not exactly implemented. Our solution is stable for ARM on X86, but is unlikely to solve all guests. Indeed even for ARM we were able to hypothesize a test case that would fail, though we have not been able to produce the true effect in reality. Nonetheless, we consider this as a negative result.

Overall, the result is that the speed improvement we get for multithread TCG is impressive. Our implementation is stable over Linux boot and all loads that have so far been run. We have not been able to show any issues from the atomic instruction implementation. For more details, see Section IV.

To sum up MTTCG and QBox, we have researched both approaches to multithreading simulation. Our conclusions so far are that multithreading QEMU itself is ideally suited to homogeneous (SMP) cores, while multiple instances of QEMU (QBox) are suitable for heterogeneous (AMP) cores.

F. Limitations

The SystemC scheduler is not preemptive, ie all processes run in cooperative mode. Each process handing control back the the kernel, under its own control. The SystemC scheduler will never interrupt a running process. It is necessary to introduce synchronization points to pass control back to the kernel. When there are no more processes to execute, then SystemC exits. In addition, the SystemC kernel is not thread safe. Since the scheduler is not preemptive, it is not really a requirement. This means that models written in SystemC do not have to consider thread safety issues. It simplifies the model and speeds up the development process. However from our point of view, this is problematic as we run several different threads for different CPUs. However, a thread safe mechanism has been added to SystemC to enable interaction between SystemC and other OS threads and processes. It is a specific thread safe event. It is the mechanism applied when QBox wants to post an event to SystemC simulation kernel.

If QBox wants to access an IO within a SystemC model, it can use a thread safe event which will then be executed by SystemC. It guarantees that the single SystemC thread is used to execute SystemC code. This solution also maintains the single threaded nature of SystemC. It adds complexity due to lock system between threads and decreases performance of simulation. In general, it is possible to write thread safe SystemC models. If the models connected to QBox were thread safe, it is not necessary to switch threads. In this case, QBox would be able to directly access IO without the synchronization mechanisms. To add this feature, we would like to add a new way to inform QBox that this part of SystemC simulation is thread safe and can be directly accessed without interruption. It has to be compliant with the TLM-2.0 standard. This will speed up simulation and improve usage of parallel threads.

G. *Open source license and business potential*

QBox, as a fork of QEMU, is a free and open source project released under GPLv2 license (see QEMU header files). QBox and the QBox libraries are also released under the GPLv2 license. TLM2C is our wrapper that can take any 'C model' based on structures like TLM-2.0 standard. It converts it into SystemC (C++) code. This is released on the GPLv2 with the additional right to use the code with SystemC. TLM2C is mealy a library that enables the construction of a SystemC model from a C based library, which could be QEMU or any other C based model.

Model constructors in an industrial setting can choose to use any TLM-2.0 standard compliant CPU models that suits their needs. We believe that QBox is an exceptionally good choice because of its quality, speed, license and features. When industrial model constructors ship their models to third parties, they do not have generally the rights to ship CPU models coming from other vendors. This is equally the case with our models. The customers must be able to select whichever CPU model suits their needs. This was the key motivating driver behind the TLM-2.0 standard. Again, whatever CPU was chosen during the initial construction of the platform, we believe that QBox is a good choice as a CPU TLM-2.0 model.

In addition to the quality and speed of a QEMU based solution, the open source license also guarantees the longevity of the solution. Indeed, as users are able to keep, modify and re-use the solution for as long as required. In the safety critical industry, this is a key advantage of open source as solutions need to be re-usable for decades into the future. Indeed, users can develop themselves of have any competent developer maintain and develop models based on this solution. This can encourage services based businesses, geographically placed in regions specializing in specific domains.

IV. Experimental results

A. *Introduction*

Experimentations are done with QBox, QEMU-SC, and ARM Versatile Express board containing a CoreTile Express A9 daughter board in addition to the motherboard. The daughter board contains four A9 cores. It is a good candidate to compare results for a multi-core SMP platform. QEMU officially supports this board.

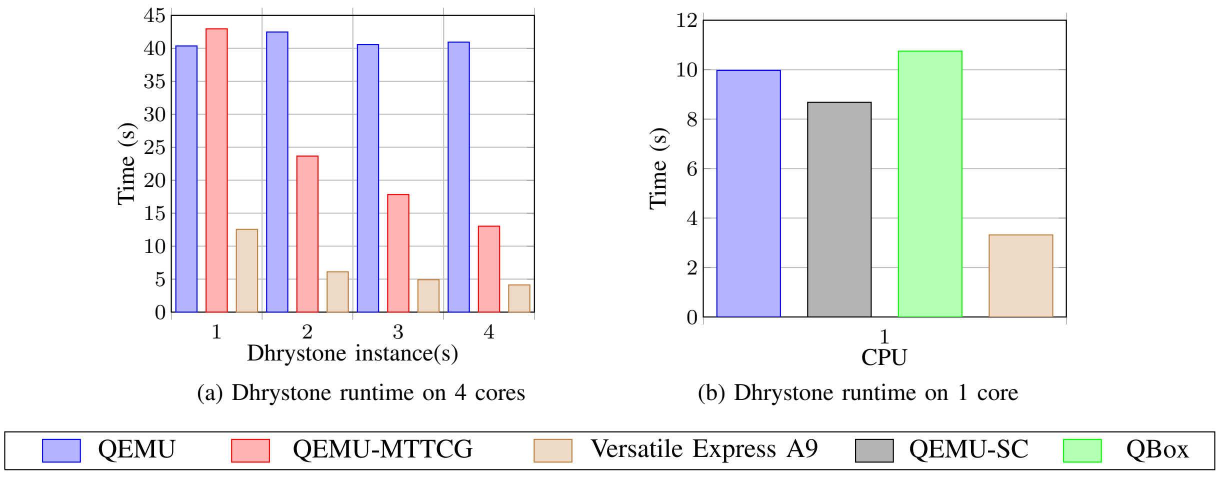
All benchmarks (see Section IV-B) have been executed on real hardware in addition to QEMU, QEMU with MTTCG, QEMU-SC and QBox on a computer running an Intel Xeon E3-1271 (at 3.6 GHz) and 32Gb of memory. We have modified the DTB (Device Binary Tree) to use only those devices that are available to all virtual and physical platforms. They are: the 4 x Cortex A9, System Controller, Generic Interrupt Controller (GIC), SP804 (Dual Timer) and PL011 (UART).

The objective of QEMU-SC is to model the majority of the platform inside QEMU by adding externally only a few elements. Specifically QEMU-SC expects that the memory will be held inside QEMU. In contract, the philosophy behind QBox is that the memory map is handled in SystemC. To keep the comparison as fair as possible, we set up the two environments as follows: QEMU-SC will run all Cortex A9 cores, System Controller, the full memory map and ARM GIC within QEMU. Both the SP804 and PL011 will be in the SystemC. QBox will run an A9 multi-core node with all 4 Cortex A9 processors, and the tightly coupled ARM GIC. The current version of QBox (1.3.0) is based on QEMU 2.3.0. ARM GIC is shared by the multi-core A9's on a private bus. The full memory map, the SP804 and PL011 will be specified in SystemC. After some experimentations, as we explain below, we have decided to fix a quantum value of 1ms for QBox and QEMU-SC.

B. *Benchmarks*

In order to provide a measure of performance focused on raw processing power, we have chose to use Dhrystone[18]. Indeed, it is readily available on all platforms. Dhrystone is an open source synthetic computing benchmark program used to measure the integer performance of processors. We used Dhrystone 2.1 that is built with the Buildroot toolchain. The program has been added to root file system to be run in the guest OS (Linux). 10^7 Dhrystone computations were benchmarked. As the time reported by a virtual machine may not be accurate, an external timer was used to measure execution time, averaging over 5 runs.

One of Dhrystone deficiencies in terms of benchmarking is a total absence of IO. In order to compare a more 'real world' scenario, we have also compared Linux boot performance which does use a mixture of IO's and processing. We built Linux kernel and a light root file system for our platform by using Buildroot 2015-08.01 with Linux kernel v4.1.4. We only enabled the required devices to boot the platform and run the benchmarks.



(a) Dhrystone runtime on 4 cores

(b) Dhrystone runtime on 1 core

Fig. 6: Dhrystone runtime on QEMU, QEMU-MTTCG, and Versatile Express A9 with different number of CPUs

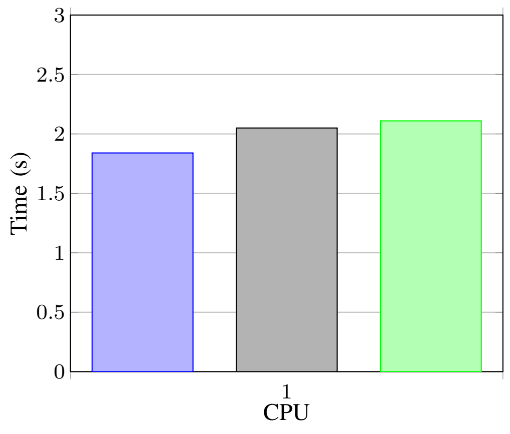


Fig. 7: Different solutions booting Linux

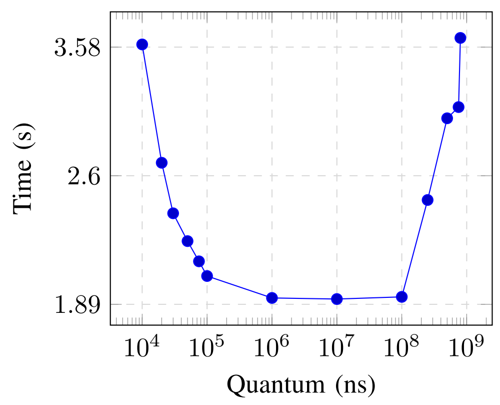


Fig. 8: Quantum impact on Linux boot using QBox

C. Dhrystone

As we can see on Figure 6, we benchmarked 4 Dhrystone's that are executed at the same time with different numbers of CPUs using real hardware, QEMU and the new QEMU-MTTCG. We suitably modified the DTB to limit the number of CPU's being used. The current QEMU does not take advantage of increasing the number of CPUs. Indeed, the time to compute the four Dhrystone is approximately the same for all configurations. However both QEMU-MTTCG and the real hardware are both capable of dividing the load over multiple CPUs. For of a single CPU, QEMU-MTTCG is slightly slower than standard QEMU. This is the overhead of ensuing thread safety in QEMU, measured as about 14%. For the case of a 4 cores CPU, QEMU-MTTCG is over 3 times faster than the existing QEMU. Overall, MultiThreaded TCG (MTTCG) has now been demonstrated with an impressive near linear speed improvement. We plan to implement MTTCG in QBox in a future work to speed up SMP CPUs simulations. Finally, we can see that QEMU emulator is around 3 to 4 times slower than real board on our host machine. Real board cores run at 1.3GHz so the host machine executes one virtual CPU instruction all around 10 host instructions.

On the Figure 6b, we can see that overall QEMU is only 3-4 times slower than a board running at 1.3GHz. This is clearly highly dependent upon the host and guest. However, this falls well within the often assumed 'one order of magnitude', even for quite complex cores such as the A9. The various SystemC solutions (QEMU-SC and QBox) are both substantially similar to QEMU in terms of CPU computation performance. QEMU-SC runs very slightly faster than QEMU in this test, this is due to the version of QEMU used in QEMU-SC. The older version of QEMU used in QEMU-SC (close to version 1.4.50) runs Dhrystone about 20% faster than the version of QEMU used in QBox (and for the other QEMU tests), which is version 2.3.0. Applying a ratio between QEMU 2.3.0 and QEMU 1.4.50, we estimate that QEMU-SC overall has a performance cost of around 3% compared to the QEMU being used. QBox is slightly slower than QEMU-SC, specifically for IO accesses as discussed above. QBox has a performance cost of 8% compared to the QEMU being used.

D. Linux

During boot step, Linux performs a few thousands IO. As shown in Figure 7, we can see that boot time of all the solutions are very close to the time set by QEMU itself. QEMU-SC and QBox are a little bit slower than the original QEMU. Even though Linux boot performs significant amounts of IO, the boot times of all three options are substantially similar. We do not see a marked slow-down in any of the solutions. Clearly the way in which Linux boots, and how it times IO activity during the boot has a significant impact on these numbers. However, we argue that far from being atypical, Linux boot is one of the typical things that the users of virtual platforms will run, time and time again (especially if they are in the process of developing low level software, the very target of virtual platforms of this type). Hence the fact that all three solutions have a Linux boot time which is substantially similar is highly relevant.

E. Quantum

As shown in Figure 8, we can see the impact of the quantum duration on Linux boot by using the QBox virtual platform. We get the smallest boot time with a quantum around 1ms. For smaller quantum durations, QBox will process IO more frequently. It increases execution time and decreases simulation speed. However, as quantum durations increase, Linux will see fewer and fewer timer interrupts, which it uses to synchronize and schedule processes. As a result, processes which are spinning waiting for others, or waiting for a timer interrupt will potentially spin for longer, slowing the overall simulation to the point at the far right hand side of the graph where Linux is no longer capable of running. As the selection of an appropriate quantum is hard, it takes experimentation, and is system dependent. Typically, a 'bath' shaped graph can be expected, users may choose a lower quantum if higher timing fidelity is required.

V. Conclusion

This article provides a review of the existing solutions to interfacing QEMU and SystemC. We go on to look more closely at two solutions: QEMU-SC and QBox. Both solutions has been compared to get an objective point of view on the best solution. Some use cases have been studied. They show that QBox is more in line with current and future requirements by supporting homogeneous and heterogeneous simulations. Different approaches are proposed to simulate performance in SystemC and specifically multi-core simulations. Multiple solutions in the article are proposed for SMP and AMP platforms. For SMP, a new multithread solution is presented for QEMU called MTTCG. MTTCG results are good. Indeed, a linear speedup is achieved. All the works detailed here are available as open source on the GreenSocs.com web site. We actively seek feedback for our research activity.

References

[1] OVP. Open virtual platform. [Online]. Available: http://www.ovpworld.org

[2] QEMU. Qemu. [Online]. Available: http://www.qemu.org

[3] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, Jan 2012.

[4] GreenSocs. Qemu-sc. [Online]. Available: http://git.greensocs.com/qemu/qemu-sc

[5] Lip6. Soclib. [Online]. Available: http://www.soclib.fr

[6] T.-C. Yeh and M.-C. Chiang, "On the interface between QEMU and SystemC for hardware modeling," in *Next-Generation Electronics (ISNE), 2010 International Symposium on*, Nov 2010, pp. 73–76.

[7] F. Cucchetto, A. Lonardi, and G. Pravadelli, "A common architecture for co-simulation of systemc models in qemu and ovp virtual platforms," in *Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on*, Oct 2014, pp. 1–6.

[8] M.-C. Chiang, T.-C. Yeh, and G.-F. Tseng, "A qemu and systemc-based cycle-accurate iss for performance estimation on soc development," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 593–606, April 2011.

[9] C.-S. Peng, L.-C. Chang, C.-H. Kuo, and B.-D. Liu, "Dual-core virtual platform with qemu and systemc," in *Next-Generation Electronics (ISNE), 2010 International Symposium on*, Nov 2010, pp. 69–72.

[10] M. Monton, J. Engblom, and M. Burton, "Checkpointing for virtual platforms and systemc-tlm," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 133–141, Jan 2013.

[11] M. Monton, A. Portero, M. Moreno, B. Martinez, and J. Carrabina, "Mixed SW/SystemC SoC Emulation Framework," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, June 2007, pp. 2338–2341.

[12] GreenSocs. Greenlib. [Online]. Available: http://git.greensocs.com/greenlib/greenlib

[13] ——. Greensocs forge. [Online]. Available: http://git.greensocs.com/explore

[14] R. D. A. D. D. K. G. Liu, T. Schmidt, in *NASCUG Meeting, 6/2/2014.*

[15] J.-H. Ding, P.-C. Chang, W.-C. Hsu, and Y.-C. Chung, "Pqemu: A parallel system emulator based on qemu," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, Dec 2011, pp. 276–283.

[16] D.-Y. Hong, J.-J. Wu, P.-C. Yew, W.-C. Hsu, C.-C. Hsu, P. Liu, C.-M. Wang, and Y.-C. Chung, "Efficient and retargetable dynamic binary translation on multicores," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 3, pp. 622–632, March 2014.

[17] Z. Wang, R. Liu, Y. Chen, X. Wu, H. Chen, W. Zhang, and B. Zang, "COREMU: a scalable and portable parallel full-system emulator," in *Proceedings of the 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2011, San Antonio, TX, USA, February 12-16, 2011*, 2011, pp. 213–222. [Online]. Available: http://doi.acm.org/10.1145/1941553.1941583

[18] ARM. Dhrystone benchmarking for arm cortex processors. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.dai0273a/DAI0273A_dhrystone_benchmarking.pdf

Session 12

Dependability

Thursday 28th, 09:00 – Ariane 2

SAFER MARINE AND OFFSHORE SOFTWARE
WITH
FORMAL-VERIFICATION-BASED GUIDELINES

Lucas Duboc, Sébastien Flanc, Florent Kirchner, Hélène Marteau, Virgile Prevosto, Franck Sadmi, Franck Védrine
{florent.kirchner, virgile.prevosto, franck.vedrine}@cea.fr
{lucas.duboc, helene.marteau, franck.sadmi}@fr.bureauveritas.com
sebastien.flanc@sirehna.com

Abstract:

As the development of ship software systems has followed the growth curve of digital technologies, Marine & Offshore assessors, like Bureau Veritas, are lacking dedicated software safety assessment standards and tools compared to other industrial sectors like railways or aeronautics. Indeed, in this field of Marine & Offshore, software systems are seen as *black-boxes*, i.e. only assessed through system testing without specific requirements for the software development. Given the impacts of software failures on human, strategic, economic, or environmental aspects, this approach is not sufficient. From this statement and because usual safety standards are very demanding in terms of development processes, Bureau Veritas has decided to issue pragmatic guidelines for the development and the assessment of industrial software. They are focusing on development processes and the use of efficient tools to verify software through a *white-box* approach. In this context Bureau Veritas has partnered with CEA-List which is a major actor in applied formal verification techniques. This paper is illustrated by a use case with Sirehna for the implementation of those guidelines on a critical ship software system.

Keywords:

Industrial software, Safety, Marine & Offshore, Software standards, Static analysis, Formal methods, White-box approach, Certificate of conformity

1. Context of the Partnership

Digital technologies have overwhelmed industrial markets, and software systems are underlying almost all technical products. Simple or complex, the behavior of any electronic equipment is directly controlled by a piece of software, whose failure can lead to severe human, strategic, economic, or environmental consequences. It is now well known and understood that software testing, because of its intricate logical nature, cannot detect all defects that might have been inserted in the software design and implementation phases. What's more, the more complex the software system is, the more costly it is to perform verification activities, and the later undetected defects may remain. Finally, once put into service, modifications are often applied to software components to provide new functionalities or correct detected errors. Such changes also bring in complexity and are potential sources of flaws within the software system. For all these reasons verification activities, providing assurance that the software system will function correctly as intended during all its life – from commissioning to decommissioning of its host equipment – have to be carried out throughout software development activities, in a fashion that is mindful of cost-effectiveness and maintainability.

1.1. Software verification in BUREAU VERITAS Marine certification schemes

In industrial sectors such as aviation, nuclear energy, and railway, where life- and safety-criticality have historically represented the main concern, advanced functional safety standards have become conditions to access the market. As safety is their prime concern, these standards are very demanding for the organizations that have to conform to their objectives. For the industrial sectors where safety aspects are less prevalent, best practices of software development are widespread without being gathered in a single document. This is the case within the marine & offshore industry, which constitutes the historical core business of BUREAU VERITAS, where the International Association of Classification Societies IACS is responsible for the establishment of standards to verify and assess ship safety. BUREAU VERITAS is one of the 12 member companies of this association, and was entrusted with its chairmanship from July 2014 to July 2015.

So far, the standard requirements related to the assessment of ship software components were generic and limited. Unified Requirements (IACS, 2010) had been dealing with high-level requirements for software assessment, typically test completion at system level. Yet experience gathered from the aforementioned best practices for critical systems has shown that a system-level testing approach is too limited – either in terms of coverage, or conversely in terms of cost-efficiency – when verifying software that implements rich functionalities and dealing with multiple inputs / outputs. Intrinsic complexity of the software system has to be taken into account and processes of development have to be adapted accordingly.

In this context, BUREAU VERITAS Marine wants to improve its software assessment scheme (based on a white-box approach) to get more confidence in software systems used in the marine & offshore industry.

1.2. CEA LIST expertise in source code verification tools

Prompted by criticality concerns, various approaches to software assessment have been investigated by the verification community. In particular Software Assessment tools focus on detailed program analysis techniques, i.e. tools and methods that provide software developers and auditors with strong and demonstrable confidence in their artefacts. Overall, these techniques allow the safety experts to verify that programs and their functionalities behave according to their specification. These techniques can be further divided into two classes:

1. The analysis of programs in a non-runtime state, called *static analysis*, proceeds across the source code just like a compiler would, looking for patterns indicative of unexpected behaviors.
2. The analysis of programs in a runtime state, called *dynamic analysis*, runs the compiled source code on sets of predefined input values to detect unexpected behaviors.

Different types of static analysis techniques can be applied to perform code assessment, some of them based on formal methods. Two main trends have emerged, both using abstractions of the code in order to formally verify particular classes of properties.

1. A first approach, symbolic execution (which broadly speaking encompasses deductive verification and model checking), uses logic solvers that relate code blocks to their expected behaviors, as expressed by specific logical assertions in the source code.
2. A second approach, called abstract interpretation, computes for each variable of the program an abstraction of the values it can take during any program execution, warning when it encounters unexpected behaviors.

Over the past decade, these types of program analyzers have been successfully applied to demonstrate the safety of life-critical systems. Program verification is used in this field to achieve demonstrably equivalent levels of safety as those attained with traditional methods for critical systems, but at a lower cost (Randimbivololona, Souyris, Baudin, Pacalet, Raguideau, & Schoen, 1999). These practices have been successfully used in the context of various domain-specific certification standards (DO-178B/C, CENELEC EN 50128, IEC 60880, IEC 61508, ISO 62304, ISO 26262, etc.).

Frama-C is a source code analysis platform for C99 ISO C programs, developed by CEA LIST and its partners. It implements both static and dynamic source code analyzers as modular *plugins* (see section 3.2). Frama-C differs from other static analyzers as it provides a diverse set of formal tools that cooperate through code annotations emitted in the ACSL language (Baudin, Filliâtre, Marché, Monate, Moy, & Prevosto, 2015).

1.3. Shared Objectives of the partnership

Both BUREAU VERITAS and CEA LIST deal everyday with some of most of the critical industrial sectors and associated standards for software development, and have a strong knowledge of the best practices in these different environments. As said in section 1.1, there currently exists little guidance when it comes to software verification in marine & offshore applications, buts plenty of experience on lifecycle certification. Furthermore, as presented in section 1.2, recent software analysis techniques have demonstrated their relevance in software verification for product assessment schemes.

It is with this dual assessment methodology in mind – process oriented verification and source code analysis – that BUREAU VERITAS and CEA LIST partnership has been working on SOFTWARE DEVELOPMENT & ASSESSMENT GUIDELINES. This document, whose salient features are presented in section 2, provides minimum objectives that should be met by any software system that needs to demonstrate its ability to achieve an expected performance level, thus satisfying common safety concerns.

This document was designed to be usable in all the domains that deal with the quality and safety of software systems. On the one hand, it had to be seen as guidelines of development and did not aim to supersede existing safety standards that are regulatory in some specific industries like railway or aeronautics. On the other hand, those guidelines had to be fit to constitute the reference standard used by BUREAU VERITAS when assessing software systems in marine & offshore in order to establish confidence in terms of safety. The guidelines were designed to be relevant whatever the programming language used for the software system development. No programming language was favored as long as their dangerous or complex instructions are identified and controlled. Even if the guidelines recognized the use of tools, no specific solutions are identified: each project would choose dedicated tools depending on its constraints and objectives.

The continuing partnership between BUREAU VERITAS and CEA LIST has allowed the team to successfully share knowledge and expertise, in particular on safety systems assessment and certification, and software verification.

2. Originality of the software assessment methodology

2.1. Guidelines rationale

Simplicity was a major objective during the writing of the SOFTWARE DEVELOPMENT & ASSESSMENT GUIDELINES. The authors strived to define a practical methodology that minimizes both efforts of software development and software assessment. The small size of the document (with a goal fixed at a maximum of 50 pages from the beginning of the development of these guidelines) is designed to ease its use and circulation.

Even though it does not attempt at a thorough comparison between the main software safety-related standards that are enumerated in section 1, such as the work of (Ledinot, et al., 2014), the document follows the same philosophy and structure as these various standards. The main idea in software development is to keep in mind that quality and safety are properties progressively achieved by construction, following a process to avoid introduction of errors, to be tolerant with errors and to eliminate errors in the software system.

In terms of philosophy, the document is grounded on the assumption that software developments are framed by well-defined and verified processes which are structured following a standard V-lifecycle. In terms of structure, the software development methodology is composed of four main parts. The first step of the methodology consists in a risk analysis to identify the criticality level related to the computer-based system that hosts the software system. This criticality level allows the categorization of the software system in the same manner as its targeted performance level. The second part focuses on the choice and qualification of COTS and software support tools as it is an important topic which is too often underestimated in many industrial sectors. The third part deals with classic quality insurance aspects like project, configuration, change & documentation management. The fourth part relates to the development activities, from specification to installation, and their associated acceptance criteria and verification milestones. In addition to these four main parts, annexes describe the link between the risk classification approaches of other main safety-related standards and the criticality scale of the guidelines. They also propose an application of the guidelines objectives to model-based developments.

In order to assist software development and assessment teams, the document contains tagged key objectives along with their applicability for the targeted software performance level. The development and assessment processes of a given software system can be tailored to comply with a limited subset of objectives of the guidelines, depending on the performance level of the software system. Here is an example of one of these guidelines objectives:

```
OBJ_TOOLS_010
Each SOFTWARE SUPPORT TOOL used at any step of the SOFTWARE SYSTEM development shall
be identified.
```
`[I, II, III, IV]`

Here, the objective is required when developing SOFTWARE SYSTEM of performance level 1, 2, 3 or 4 (i.e. all the potential performance levels defined in the guidelines). Finally, in order to ease the software assessment work, the document offers an assessment checklist that lists in a table each guidelines objective with its applicability, and provides a “justification” field to state if it has been fulfilled or not by the development team.

Nevertheless, the main added value of the SOFTWARE DEVELOPMENT & ASSESSMENT GUIDELINES stands in the innovative positioning of code analysis. Formal techniques are proposed here as solutions for functional and non-functional verifications, including computation accuracy, parameter integrity, and behavioral conformance. While dynamic testing evaluates if the software system behaves as intended in particular in interaction with hardware and environment, static analysis assesses whether the software system is built correctly or not, thus levelling the confidence in its ability to behave correctly. Therefore, without minimizing the necessity to conduct verification activities to improve the confidence in the ability of the software development process to produce the right software system, the SOFTWARE DEVELOPMENT & ASSESSMENT GUIDELINES stress the part played by static analysis techniques at the software testing stage to enhance the demonstration of the behavioral correctness of the software system. This proposal can be linked to recent approaches that advocate moving from process-based towards more assurance-based certification process, especially in the aeronautics domain (Rushby, 2009) (Holloway, 2013).

The following two objectives taken from the guidelines illustrate this approach:

```
OBJ_DEV_185
Dead code and run-time errors shall be detected. Recursivity is accepted only if
controlled by a maximum number of iterations.
```
[II, III, IV]

```
NOTE1: Depending of the programming languages, typical run-time errors are:
```
- Buffer overflows (underflow);
- Out of bound accesses (arrays, pointers, …) and null pointer dereferences;
- Invalid arithmetic operations (division by zero, square root of negative numbers,);
- Non initialized variables access;
- Dangerous type conversions;
- Shift in bitwise operations with a negative value or a value greater or equal than the size of the underlying type (C/C++); etc…

```
These kinds of errors are occurring during the execution of the SOFTWARE SYSTEM; even
if they are syntactically correct.
```

```
OBJ_DEV_205
Code analysis tools shall be used to check dead code and run-time errors.
```
[III, IV]

2.2. Comparison with other standards

The guidelines are built following common functional safety standards, defining a rigorous development lifecycle and focusing on the use of tools to automate some tasks and improve some verification activities. Four key issues can be identified to compare the guidelines with the well-known IEC 61508 (61508, 2010): criticality-dependent rigor, verification coverage, organization and verification of verification.

The first key issue addresses the efforts to produce during software development that depend on its severity or criticality. This is a common point with the IEC 61508 dealing with SIL as the guidelines keep the idea of a scale of development efforts relating to the expected performance level from 1 to 4, 4 being the highest. This performance level is an input when developing a software system, it has to be determined during the system analysis where all risks are analyzed.

Concerning the verification coverage, some differences exist between the two documents. When checking the detailed design, IEC 61508 defines specific structural coverages like MC/DC, path coverage, etc. whereas the guidelines only focus on functional coverage. This is offset by the strong verification efforts asked on the source code by code analysis tools. This choice has been made because structural coverage can prove to be genuinely time-consuming, even if necessary, so the focus has been placed on functional and source code analysis. Moreover, the classical "software module testing" of the IEC 61508 has been called "checking of the software units" and it explicitly includes either the dynamic testing activity or the use of source code static analysis.

Regarding the development team organization, and especially the independence required to carry some activities, the guidelines ask for independence between the left and the right sides of the V-cycle and also for all verification activities. Nevertheless, these independence objectives do not freeze any team configuration (they can be achieved by peer-review in a same development team). This allows more flexibility for small development teams. The IEC 61508 is not really precise either on this specific topic; the only clear requirement is about the independence of assessors.

While the generic functional safety standard for electronic devices IEC 61508 requires tool qualification, the guidelines moreover recommend a usefulness analysis. It has to be done at the beginning of the software development, and it aims to evaluate pros and cons of performing manual or automated activities.

Last point concerns the verification of verification, like in IEC 61508 all activities have to be verified and eventually assessed. In case of third part assessment, a certificate of compliance may be issued regarding the guidelines.

3. Application of the methodology on a use case

3.1. SIREHNA problematic

SIREHNA is a subsidiary of DCNS Group, and a part of the technological research center DCNS Research. SIREHNA features different fields of expertise such as hydrodynamics and control of mobile maritime units.

Those skills are directly applied to maritime embedded systems such as Dynamic Positioning Systems; Frigates Rudder Roll stabilization systems, Unmanned surface Vehicles or critical tailored systems such as the French aircraft carrier flight deck tranquilization system. Those embedded systems include software components which perform many functions such as real-time computation of navigation data, multi-degree of freedom control algorithms, actuators orders and supervision of the system by human operators. These systems embedded on military ships, submarines, and offshore vessels operating near oil platforms, are safety critical and their complexity requires a rigorous development, verification and validation process.

Today, IACS does not impose code-level standards in the naval environment for the development of critical software applications. For example, the generic safety standard IEC 61508 is not specialized to the maritime domain.

BUREAU VERITAS and CEA LIST aim to provide a formal framework defining verification objectives and milestones during software development. SIREHNA supports this project by providing its industrial vision both in terms of system features and operation, as well as process and industrial constraints for their development. The goal is to implement and apply the tools proposed through BUREAU VERITAS & CEA LIST initiative to ensure that they are relevant to industrial constraints and assess their performance in a concrete development context.

In this context the work done with BUREAU VERITAS and CEA LIST enables SIREHNA to anticipate new standards dedicated to the naval field.

3.2. Overview of Frama-C and Fluctuat

Two main tools have been used during this proof of concept: Frama-C (Kirchner, Kosmatov, Prevosto, Signoles, & Yakobowski, 2015) and Fluctuat (Delmas, Goubault, Putot, Souyris, Tekkal, & Védrine, 2009), which are both mainly developed at CEA LIST.

Frama-C is an Open-Source (LGPL-licensed) framework dedicated to the analysis of C programs. It is built around a kernel tasked with the parsing and type-checking of C code and accompanying ACSL annotations if any, as well as maintaining the state of the current analysis project. This includes in particular registering the status (valid or invalid) of all ACSL annotations, either user-defined or generated, emitted by the various analyzers. Analyses themselves are performed by various plugins, that can validate (or invalidate) annotations, but also emit hypotheses that may eventually be discharged by another plugin. This mechanism allows some form of collaboration between the various analyzers. Many plugins exist, but in the remainder of this section we focus only on the ones that have been used during the case study on the application of the proposed guidelines over Sirehna's code.

While Frama-C's kernel is meant to operate on C programs, the case study has fueled the development of a plugin for analyzing C++ code. This plugin, named frama-clang, whose prototype has been elaborated during the European FP7 project STANCE[1] uses the clang compiler as a front-end for type-checking C++ code and converts clang's abstract syntax tree (AST) into Frama-C's own AST, producing C code equivalent to the original C++ program. In addition, frama-clang extends clang with ACSL++ annotations that are converted into ACSL annotation together with the translation of the code.

Two important analysis plugins are Value Analysis and WP. Value analysis is based on abstract interpretation, and computes an over-approximation of the values that each memory location can take at each program point. When evaluating an expression, Value Analysis will then checks whether the abstraction obtained for the operand represents any value that would lead to a runtime error. For instance, when dereferencing a pointer, the corresponding abstract set of location should not include `NULL`. If this is the case, Value Analysis will emit an alarm, and attempt to reduce the abstract value. In our example, it will thus remove `NULL`. The analysis is correct, in the sense that if no alarm is emitted, no runtime error can occur in a concrete execution. It is however

[1] http://stance-project.eu/ and http://llvm.org/devmtg/2014-04/PDFs/Posters/FramaC.pdf

incomplete, in the sense that some alarms might be due to the over-approximations that have been done and might not correspond to any concrete execution. Various settings can be done to choose the appropriate trade-off between the precision and the cost of the analysis. While the most immediate use for Value Analysis is to check for the absence of runtime error, it will also attempt to evaluate any ACSL annotation it encounters during an abstract run. Such verification is however inherently limited to properties that fit within the abstract values manipulated by Value Analysis. Mainly, it is possible to check for bounds of variables at particular program points.

WP is a deductive verification-based plugin. Unlike Value Analysis, which performs a complete abstract execution from the given entry point, WP operates function by function, on a more modular basis. However, this requires that all functions of interest as well as their callees be given an appropriate ACSL *contract*. Similarly, all loops must have corresponding *loop invariants*. When this annotation work has been completed, WP can take a function contract and the corresponding implementation to generate a set of *proof obligations*, logic formulas whose validity entails the correction of the implementation with respect to the contract. WP then simplifies these formulas, and sends them to external automated theorem provers or interactive proof assistants to complete the verification. WP's main task is thus to verify functional properties of programs, once they have been expressed as ACSL annotations. It is however also possible to use it to check that the pre-conditions written for a given function `f` imply that no runtime error can occur during the execution of `f`.

Two other plugins mentioned in the rest of the paper are not analyzers *per se*, but can help the main analysis plugins in performing some verification task on complex code. First, the Slicing plugin performs program transformations. More precisely, given a *slicing criterion*, e.g. the values of some variables at a given program point, Slicing will remove all instructions of the original program that do not contribute to this criterion. The result is a simpler program, which is equivalent to the original one with respect to the criterion. Slicing can thus be used as a front-end to other, more specific, analysis when focusing on a given property of interest. Second, the EACSL plugin transforms ACSL annotations into C code with `assert` in order to allow for dynamic runtime checks. It can for instance be used in complement of Value Analysis to evaluate during the execution of unit tests whether an alarm emitted by Value Analysis might occur in practice. Similarly, if a given ACSL annotation cannot be discharged by WP, it is still possible to check whether it holds during concrete executions thanks to EACSL. Note however that EACSL only supports a subset of ACSL, and that the instrumentation it performs might have an impact on the execution time of the program.

Finally, Fluctuat (which is not a Frama-C plugin but benefits from the Frama-C toolchain) focuses on the accuracy of floating-point computations. It is based on abstract interpretation and computes an over-approximation of the magnitude of the error due to rounding when performing a sequence of floating-point operations. Given a mathematical specification of what the operations are supposed to compute, it can also indicate the magnitude of the error due to the method used (e.g. Newton algorithm for extracting a square root). The contribution of each individual floating operation to the overall rounding error can also be traced, in order to help designing more robust algorithms if needed.

3.3. Sirehna's code analysis by Frama-C

From a methodological point of view, the aim of this code analysis activity is to prepare the certification. Formal proof on the code gives confidence in the library and ensures that the expressed properties are under control. The application of the development guide extends the current development process followed by SIREHNA to develop its applicative software. This process is based on the development of internal libraries that are likely to be shared, customized and then embedded in the applicative software. Current libraries come with unit tests with single input, validation unit tests with multiple inputs, non-regression tests. Current applicative software also comes with integration tests and non-regression tests. All these tests are good starting points for defining the verification scenarios that will drive the abstract interpretation based analyses. The first step just consists in replacing the input values of the tests by some ranges of values.

For the verification process of the numerical libraries, SIREHNA, CEA LIST and BUREAU VERITAS have decided to experiment a bottom-up verification approach. It benefits from the formal analysis tools WP and Fluctuat that are initially designed for the unitary verification of single components. At component level, some short verification cycles like annotation/analysis/result examination aim to deliver proved annotations in the source code. The annotations carry information about the domains, the loop invariants and the accuracy of the computations. The assembly of components in the library come with an assembly of the annotations guided by the user. The objective is for high level annotations to meet the properties exported by the functions of the library, as verified through whole program analysis.

Unitary analysis

Unitary testing, although an important step in the validation of critical systems, offers only statistical verification aligned with the coverage rate of tests: indeed, it is very difficult to cover the entire range of variation of the inputs of a function, even adopting heavily codified development processes such as DO178. Therefore, the advantage of the static analysis tools proposed by CEA LIST is to prove, for instance, the absence of certain classes of errors, or some functional properties expressed as ACSL annotations and to complement unit and functional tests. This provides an increased level of robustness and greater efficiency in error detection.

The teams of the CEA LIST performed a code analysis of various "core" functions developed by SIREHNA in C++, using thus frama-clang as front-end. An initial analysis was carried out on the source code of the ship trajectory generation functions. In a first case study, Frama-C's Value Analysis plugin was used to demonstrate its intervals values verification capabilities on floating point numbers. More precisely, the entry point for the analysis stemmed from an existing unitary test that was meant to check that the functions for converting Cartesian coordinates to polar ones and vice-versa were indeed inverse to each other (modulo rounding). For the analysis with Frama-C, we replaced floating-point inputs with small intervals around the original input, in order to check whether the function under test was robust to numerical imprecision. The entry point for Value Analysis was thus along the following lines:

```
void test(void) {
    double x = Frama_C_interval(x_0 - eps, x_0 + eps);
    double y = Frama_C_interval(y_0 - eps, y_0 + eps);
    double z = Frama_C_interval(z_0 - eps, z_0 + eps);
    double x_conv, y_conv, z_conv;
    double lon, lat, height;
    cartesian2polar(x,y,z,&lon,&lat,&height);
    polar2cartesian(lon,lat,height,&x_conv,&y_conv,&z_conv);
}
```

`Frama_C_interval` is a built-in function of Value Analysis that returns any number between the bounds given as argument, while `x_0, y_0,` and `z_0` represent the original test input and `eps` the (small) interval of variation we want to examine. Value Analysis was complemented in this task by Fluctuat.

A second analysis focused on source code developed in C language intended to be incorporated on board submarines for the weight balancing check function. This time, Frama-C's Slicing plugin was used as a front-end to let the Fluctuat tool concentrate on the analysis of the major contributors to numerical errors without undue interference with unrelated computations. Other plugins of interest for Fluctuat are frama-clang (C++ to C translator) and the constant propagation plugin (especially on the values of pointers). Finally, some preliminary experiments have been done to use WP for verifying accuracy of floating point computations. This work follows the methodology devised in an earlier collaboration with NASA (Goodloe, Muñoz, Kirchner, & Correnson, 2013).

Integration analysis

Value Analysis run at system level will then benefit from some invariants and some assertions put by the user and verified at component level. At the certification/system level the objective of Value is to prove the absence of run-time errors (dangling pointers, divisions by zero and arguments of functions like `asin` outside the interval `[-1,1]`).

Let us consider the following piece of code as a short example:,

```
/* basic low level function of the library ; LIBRARY-level */
double distance(double x, double y)
  { return sqrt(x*x + y*y); }

/* intermediate function of the library ; LIBRARY-level */
double angle(double x, double y)
  { double dist = distance(x,y);
    if (dist > MINIMAL_DIST) {
      double result;
      if (y > dist*EPSILON || y < -dist*EPSILON)
        result = atan(x/y);
      else
        result = asin(y/dist);
      if (x < 0)
        { if (y < 0) result -= PI; else result += PI; }
      return result;
    };
    return 0.0;
  }

/* elaborated high level function of the library ; LIBRARY-level */
void cartesian2polar(double x, double y, double* rho, double* theta) {
  *rho = distance(x, y);
  *theta = angle(x, y);
}

/* applicative software ; SYSTEM-level */
int main() {
  double x, y, rho, theta;
  ...
  while (...) {
    /* domain information for the applicative software */
    x = Frama_C_interval(-10.0, +10.0);
    y = Frama_C_interval(-10.0, +10.0);
    cartesian2polar(x, y, &rho, &theta);
  }
  ...
}
```

One problem faced by the certification process on such kind of library/system decomposition is that it may be non-conclusive. On the one hand, WP/Fluctuat cannot prove some properties like the result accuracy at the library level since it depends on the domains only provided at system level. On the other hand, Value cannot prove at system level that `y/dist` as the `asin` argument is in the interval `[-1, 1]` due to its internal logic targeted to be efficient on Run-Time Errors but not precise enough to know that `abs(y) <= sqrt(x*x+y*y)` without any domain information. Thus, a collaboration between the various tools is needed to achieve a complete verification. More precisely, the following verification cycle can be proposed.

1. Annotate the library with ACSL contracts and use deductive verification – WP tool, with the Gappa (de Dinechin, Quirin Lauter, & Melquiond, 2011) theorem prover that has a built-in understanding of floating-point operations.
2. Define unitary scenarios (possibly based on the existing unit tests).
3. Use Abstract Interpretation at component level with the Fluctuat tool.
4. Adjust coefficients in annotation formulas following the results of the unitary analysis.
5. Assemble the various components
6. Use Abstract Interpretation at system level with the Value Analysis plugin of Frama-C and checks that the functions of the libraries are called with arguments in the appropriate bounds.
7. If needed, use the EACSL plugin with validation unit tests and/or with integration tests for the remaining alarms in order to classify them as true or maybe false alarms.

Case studies results

Following the CEA LIST presentations, SIREHNA evaluated the following Frama-C key functions:

- Visualization of the impact of floating point operations on numerical precision, which can identify the calculations that contribute the most to the numerical error.
- The generation of ACSL specification from an existing code, giving a synthetic vision of functions and allowing to apprehend unexpected side effects and trace dependencies between variables.
- Runtime error detection such as `NaN`, buffer overflow or uninitialized tables.

SIREHNA foresees strong added value in the following Frama-C features:

- Fully formalize the contract of simple functions directly on the source code when possible
- For more complex functions, formalize the limitations of application (preconditions)

These two features will improve the source code documentation (a step toward literate programming). As these assertions can be formally verified, this will have the following impacts on design process:

- Eliminate unit test and runtime assertions when the correctness of the implementation with respect to the contract can be proven
- Put the focus on potential safety issues that cannot be formally verified, and which will require more efforts (unit tests and/or runtime assertions)

For the application specific functions, it will be possible to specify the range of various quantities (like ship altitude, latitude and longitude). This is often a pre-requisite for Value Analysis and Fluctuat.

The outcome of this work demonstrates significant interest in productivity and securing the development process. SIREHNA has started a reflection aimed at integrating Frama-C in the development processes of its system's critical functions. Besides the extension of Frama-C to C++ is followed with great interest.

3.4. Review and application of the guidelines

SIREHNA conducted a review of the development guide "SOFTWARE DEVELOPMENT & ASSESSMENT GUIDELINES."

This document is characterized by its pragmatic approach:

1. Recommendations are concise and written in the form of objectives.
2. A global approach is described and addresses the complete development cycle, project management, use of COTS, development tools.
3. The definition of software categories in relation to their criticality is declined to different processes within one single document.

In addition, the document recalls the various existing standards on the classification of risks and software. This approach by category is directly transposable in SIREHNA processes for software development with different levels of criticality.

BUREAU VERITAS, CEA LIST and SIREHNA worked on an actual project to link the recommendations contained in the guide with the development process followed on this project (the product line of Dynamic Positioning systems). The analysis generated feedback on the application of the software development guidelines and confirmed the applicability in real life development process.
This analysis has been done during the year 2015. SIREHNA has delivered the whole documentation of a Dynamic positioning system to BUREAU VERITAS. Considering that SIREHNA has been developing software for a long time, their process of development is matured enough to assess it. BUREAU VERITAS worked as a third party assessor filling the objectives matrix based on the SIREHNA documentation.

The first activity consisted in selecting the Software Category (performance level or SIL) of the software. In the IACS requirements, the systems whose failures could immediately lead to dangerous situations are Category 3 (highest category). In the annex of the guidelines, a correspondence is given between IACS Categories & Guidelines Software Categories. Thus, the Dynamic Positioning system has been rated Software Category 3 (Software Category 4 being the highest class in the guidelines). All the objectives of the guidelines have been browsed knowing that the software system needs to reach Software Category 3.
The main conclusion of this assessment is that the SIREHNA development process is compliant with the guidelines.
Nevertheless the guidelines objectives recommend to qualify more formally tools & COTS. The functional coverage of the Software System is achieved by a strong set of validation test cases (including nominal & robustness cases). Some tests cases were partially modified to check the behaviour of the software out of his bounds, especially because the DP operates depending on numerical computations with a certain accuracy acceptance.

This analysis generated feedback on the application of the software development guidelines (some objectives have been re-worded or added) and confirmed its applicability in real life development process.
The analysis will be carried on during 2016 to assess a new version of the DP. This will permit to deliver a certificate of compliance on the DP software.

4. Conclusion and Perspectives

In this paper we present three new results: the elaboration of a set of guidelines for software development and assessment, guided by marine and naval considerations; the relationship between these guidelines and existing, state-of-the-art source code analysis tools; and the application of both guidelines and tools to an industrial use case. These results are leading the current trend toward software validation in the marine and offshore industry to deal with possible accidents and optimize operational uptime. The approach they advocate has shown potential benefits to manufacturers and end-users of the domain, and upcoming experiments will further investigate the impact of this approach on the overall software development process. Finally, while they were developed with a specific application field in mind, the guidelines presented here are generic enough that they could be applied to numerous other industrial fields where safety is a quickly emerging concern. The guidelines are freely available on the web site of BUREAU VERITAS (http://www.bureauveritas.com/home/about-us/our-business/industry-offer/software).

5. References

61508, I. (2010). *Functional safety of electrical/electronic/programmable electronic safety-related systems.*

Antoine, C., Trotin, A., Baudin, P., Collard, J., & Raguideau, J. (1994). CAVEAT: a Formal Proof Tool to Validate Software. *Convention on Nuclear Safety.*

Baudin, P. B., Filliâtre, J.-C., Marché, C., Monate, B., Moy, Y., & Prevosto, V. (2015). *ACSL: ANSI/ISO C Specification Language version 1.9.*

Beckert, B., Hähnle, R., & Schmitt, P. (2007). *Verification of Object-Oriented Software: The KeY Approach.*

Canet, G., Cuoq, P., & Monate, B. (2009). A Value Analysis for C Programs. *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation.*

Cousot, P., & all. (2011). *The Astree Static Analyser*. (J. F. R. Cousot, Producteur) Récupéré sur http://www.astree.ens.fr

de Dinechin, F., Quirin Lauter, C., & Melquiond, G. (2011). Certifying the Floating-Point Implementation of an Elementary Function Using Gappa. *IEEE Trans. Computers 60(2)*, (pp. 242-253).

Delmas, D., Goubault, É., Putot, S., Souyris, J., Tekkal, K., & Védrine, F. (2009). Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software. *FMICS. 5825*, pp. 53-69. LNCS.

Goodloe, A., Muñoz, C. A., Kirchner, F. K., & Correnson, L. (2013). Verification of Numerical Programs: From Real Numbers to Floating Point Numbers. *NASA. 7871*, pp. 441-446. LNCS.

Hoare, C. (1969, October). An axiomatic basis for Computer Programming. *n axiomatic basis for Computer Programming, 12*(10), 576-583.

Holloway, M. (2013). Making the Implicit Explicit: Towards An Assurance Case for DO-178C. *ISSC.* Boston.

IACS. (2010). *E22 : On Board Use and Application of Programmable Electronic Systems.*

Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., & Yakobowski, B. (2015). Frama-C, a Software Analysis Perspective. *Formal Aspects of Computing, 27*(3), 573-609.

Ledinot, E., Blanquart, J.-P., Astruc, J.-M., Baufreton, P., Boulanger, P., Comar, C., et al. (2014). Joint use of static and dynamic software verification techniques: a cross-domain view in safety critical system industries. *ERTS*, (pp. 592-601).

Marché, C., & Filliâtre, J.-C. (2007). The Why/Krakatoa/Caduceus Platform for Deductive Program. *19th International Conference on Computer Aided Verification.*

Randimbivololona, F., Souyris, J., Baudin, P., Pacalet, A., Raguideau, J., & Schoen, D. (1999). Applying Formal Proof Techniques to Avionics Software: A Pragmatic Approach. *FM. 1709*, pp. 1798-1815. Springer.

Régis-Gianas, Y., & Pottier, F. (2008). A Hoare logic for call-by-value functional programs. *Ninth International Conference on Mathematics of Program Construction.*

Reynolds, J. (2002). Separation Logic: A Logic for Shared Mutable Data Strutures. *17th IEEE Symposium on Logic in Computer Science.*

Rushby, J. (2009). Software Verification and System Assurance. *SEFM* (pp. 3-10). IEEE.

Sotin, P., Jeannet, B., & Rival, X. (2010). Concrete Memory Models for Shape Analysis. *International Workshop on Numerical And Symbolic Abstract Domains.*

Development of a safe CPS component: the hybrid parachute, a remote termination add-on improving safety of UAS

Laurent Ciarletta[1], Loïc Fejoz[2], Adrien Guenard[3] and Nicolas Navet[4]

[1]Loria, MinesNancy, University of Lorraine, INRIA project-team Madynes, France
[2]RealTime-at-Work, France
[3]ALERION, France
[4]University of Luxembourg, Luxembourg

Abstract

The use of Unmanned Aerial Systems (UAS) can be leveraged in many application domains ranging from agriculture to industry, opening up a wealth of new possibilities. However, UAS obviously raise important safety concerns and the use of the techniques, processes and standards developed for the aeronautic industry is not a feasible solution for most UAS. There is a need to bring in novel and pragmatic solutions to develop provably safe UAS in a time and cost-affordable manner. This paper reports on the development of a smart parachute which provides a safe-crash (termination) solution for UAS, one of the core safety requirements which can be complemented by other safety components in an incremental manner. The requirements elicitation phase, the design and partial verification of the termination system has been carried out using CPAL, a lightweight model-based design environment for embedded systems. The study illustrates on a specific requirement of the system how simulation and fault-injection on models can be used to provide evidence that the parachute system meets its design objectives.

1 Introduction

1.1 Context of the study

Drones or Unmanned Aerial Vehicles (UAV) or Systems (UAS) have been increasingly spotted on the civilian radars. Everywhere on the news, they can be seen as business opportunities in many fields from agriculture to industry, as pure entertainment devices (coming from the RC world) or as ethical and sociological subjects of interest or concerns (automated aircrafts or vehicles can be used to carry weapons, or as privacy invading tools). Having hundreds or thousands of mostly autonomous UAVs flying in rural but also urban airspaces raises important safety concerns (see [13]). One the one hand, applying or enforcing the exact same techniques and guidelines used in the aeronautic industry for the certification of small to middle size (and weight) UAVs is not a reasonable solution as of today. But on the other hand, if an RPAS (Remotely Piloted Aircraft System) / UAV industry emerges where such vehicles become ubiquitous as forecast, safety becomes a main concern for the provider of hardware and software, the operator of the system, the insurance companies or the regulation bodies. Indeed, the actual cost of certification, if applied to every part, every modification for every UAV, cannot deliver solutions even for professional systems that cost only a few thousands of euros, and could potentially be rapidly modified and updated in their uses and configurations. At least this is what is targeted for general public and professional use.

We believe that this is an opportunity to bring novel and pragmatic solutions by incrementally adding safety to the system. Beginning with a smaller set of safety functions (termination for example), and targeted subsystems the industry and legislator could increasingly extend the safety requirement to all functions and parts of the entire UAS. ALERION is promoting a design framework to build adaptive and tailor-made UAS by the integration of (secure and) provably safe Cyber Physical components. Building on the experience of the participation to the final of the UAV Outback "Search and Rescue" challenge (see `http://uavchallenge.org/search-and-rescue/`), which implied the fulfillment of a set of safety requirements, ALERION is developing in partnership with RTaW a Smart Hybrid Parachute system. This system is an all in one (*i.e.*, hardware and software) add-on termination system for any UAV. It works independently of the UAV's normal operation and can be triggered either by the operator through a safe, secure and dedicated communication channel or upon the detection of specific error conditions (e.g., hardware or software failures, system out of the authorized flight envelope, communication problems).

1.2 Contribution of this study

The main goal of this study is to demonstrate that such safety components can be developed according to a safety process in a time and cost-affordable manner, and that they would add the minimum level of safety requirements to allow a safe-crash (termination) solution for UAS. We describe design decisions and return of experience throughout the development process, with a focus on requirements and design phases.

The requirements elicitation phase, the design, simulation, and verification of the termination system has been carried out using CPAL [9], a lightweight model-based design environment for embedded systems jointly developed by RTaW and the University of Luxembourg. The complete set of requirements, the CPAL development environment (see `http://www.designcps.com`) and the CPAL models are made freely available for all uses. Though the verification stage is by no means complete, the first tests carried out using simulation and fault-injection on models suggest that the parachute system can meet its design requirements and provide a cost-effective solution to increase the safety of UAVs.

Finally, we would like to emphasize that if all the classical requirements from aeronautics should be considered, UAVs offer an opportunity to streamline the process of verification without giving away too many of the good practices of today's aeronautical industry.

2 Smart Parachute System : Remote Safety for Autonomous Vehicle Application

Very promising novel applications and markets are foreseen with UAS. But as far as micro drones are concerned, it is not possible today to enforce the same quality and safety mechanism as in private or commercial aviation[1]. Paying even a few tens of thousands euros to certify an autopilot, each and every time a new version is rolled out, can hardly work for devices that cost between a hundred and a few thousands euros. These systems are however already able to carry significant loads and represent a physical danger to the population or the institutional and industrial infrastructures.

Fully autonomous systems are not allowed so far and a human must remain in control during the operations. For now, the regulation tends to focus on the responsibility of the drone sector actors with a good part on the operator or pilot, backed by its employer. This is why in Europe it is mainly referred as RPAS (Remotely Piloted Aircraft Systems). But the systems are a lot closer to improved RC systems than to downsized aircrafts. The recreational use is now also under scrutiny. The RC world used to be a microcosm of very dedicated fans, while the "drone" is more of a mainstream activity. Very importantly, it is not clear who will be responsible in case of an accident: many non-certified hardware and software failures could potentially be invoked and it would be difficult to pinpoint the exact cause of the problem. Eventually, more autonomous UAS will require achieving and demonstrating higher safety levels.

2.1 Minimal safety for general public UAVs with our safe termination system

When dealing with autonomous systems, be they robots or autonomous vehicle, it is most often required to add some "emergency shutdown" mechanism, typically called the "red button". In case of UAS, we propose to have a safe-crash vision of safety: being able to terminate a flight when some errors conditions are observed (control loss, immediate danger etc.). This has been well stated for example in the rules of the UAV challenge "Outback Joe" [3]. The simplest of the requirements is therefore to have a termination procedure enforced on the UAV which can be triggered by the operator. Another general requirement is to have the termination procedure engaged when for example the radio contact is lost with the operator / pilot. This is a way to ensure no "out of sight" flight.

[1]The currently ongoing CAP 2018 FUI French collaborative project aims at developing the first autopilot for autonomous drones that can be certified according to DO178-C, see [14]. To the best of our knowledge, there is at the time of writing no publicly available outcome of the project.

Generally with modern autopilots for UAVs you can program that type of behavior and have some switches on your remote that actually triggers the termination or at least the return to base procedure. Most of the autopilots are however either black boxes or open source software that are constantly evolving with new features but are rarely checked as far as software correctness is concerned. Therefore, there is a high probability that native safety "mechanisms" will not be available in software in case of errors at runtime. This is the reason why ALERION has decided to develop a Smart Parachute System that works as an add-on to any already existing system. This Smart Parachute constitutes the core safety mechanism of the UAV and it has to be developed with strong safety requirements to ensure safe-crash.

Figure 1: From simulation to field test ("red button" prototype on the right).

At ALERION, we have been using several ways of providing a certain level of quality with our development tool-chain. We are using modeling environment, simulation, hardware-in-the-loop and software-in-the-loop to develop, taking into account safety concerns and accurate physical models [1,2]. We combine ROS (Robot Operating System), with MAVLink (protocol) ready devices and are using open-source autopilots. Even though the underlying OS may be "real-time" (e.g., NuttX RTOS for Pixhawk), from our experience in real flights, and analysis of parts of the code, the actual autopilot function cannot always be fully trusted. Our view is that the focus of the current developments in the UAV community is more on functionalities and ease of development rather than safety of the solutions.

2.2 Smart parachute requirements: methodology and tool support

The design of critical embedded systems is usually carried out by large OEMs using well-established processes. It is also known that the certification process costs a significant fraction of the overall design costs. Yet, we believe that these processes can be scaled down, made faster, and adapted to the design of systems that are smaller and cheaper than aircrafts, satellites or power plants. One way to achieve this is through the development of the appropriate standards, such as the ISO 29110 [5] meant for entities and services with less than 25 people, inspired by the bigger ISO/IEC/IEEE 15288. The designer needs also to be provided with the tools that will guide him along the process, and help him learn and master the state-of-the-art practices.

2.2.1 Support for requirements elicitation

The successful design of a system starts from a good specification. Yet writing a complete and coherent specification is hard, and systems engineering skills require time and experience. To help designers with requirements elicitation, RTaW provides ReqLab [5], an online requirements editor conceived to be as easy to use as a spreadsheet, for instance to edit and organize requirements, but also offering the possibility of more advanced mandatory features like requirements traceability.

If, ideally, one should be able to share a model of the system as a specification, typically a SysML model, model-based specification only is not always feasible. ReqLab provides thus features to automatically generate documents from the underlying requirements by extracting text or creating diagrams.

Finally, ReqLab tries to guide the user. On purpose, in comparison to other requirements editors, ReqLab comes with less features and a reduced degree of control. For instance, the number of links for traceability is limited to refinement. Another illustration is the usage of tags to separate between what is a "real" requirement and what is a goal (as in KAOS/GORE [6]). As a result, the user has only a few concepts to understand to start using the software.

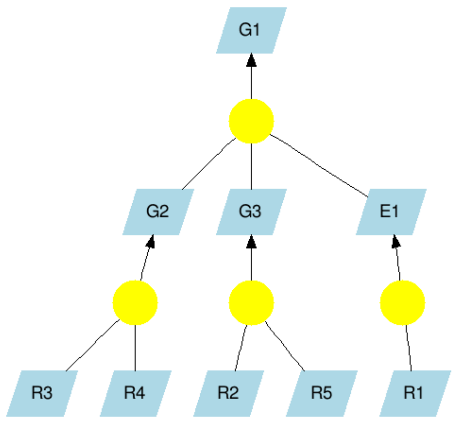


Figure 2: Generated synthesis document in KAOS/GORE notation.

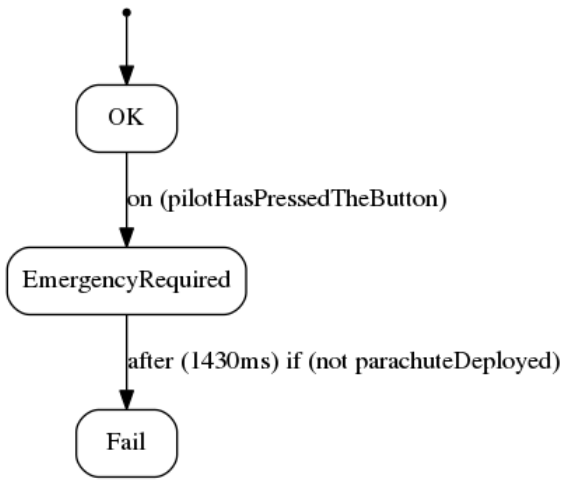
The user is encouraged to start by the mission of the system and refines it with everything that is mandatory to achieve it. Obviously those are not yet requirements per se. Hence they are tagged as "Goal". The user can write a document that extracts some of those requirements for a more understandable presentation. In the case of the Smart Parachute, the list of requirements includes:

- G1 Reduce goods damage.
- G2 Remote safety procedure shall deploy a parachute.
- G3 When communication link loss is detected, the remote safety procedure shall be engaged.
- E1 The pilot shall engage the remote safety procedure every time a hardware failure occurs, or when an emergency is going to happen.
- [R1] Every time the pilot shall be able to manually start the remote safety process.
- [R2] The remote system shall engage the remote safety process if it has received no message for 1s.
- [R3] The safety process shall turn the propellers off before deploying the parachute.
- [R4] Once the safety process engaged, the parachute shall be deployed in less that 1.43s.
- [R5] The ground control system shall send a message to the remote system every 100ms.

The value 1.43 second in requirement R4 has to be derived from a set of parameters such as the ground speed target, the minimum acceptable flight altitude, the weight of the UAS and the characteristics of the parachute (opening time, lift, etc). The maximum speed with with the UAS may hit the ground may be specified by local regulation documents (see [12] for France). A more complete list of requirements for the smart parachute is available at `https://www.requirements.fr/api/v1/docfrags/ertss2016-parachute~specification`.

2.2.2 Executable requirements

The designer can then refine deeper his list of requirements down to a specification, *i.e.* a list of requirements that are SMART (Specific, Measurable, Assignable, Relevant/Realistic and Testable/Time-bound). The fulfillment of SMART requirements can be verified in a dedicated CPAL task. CPAL natively support finite state machines, more precisely mode-automata [8], to describe the logic of the tasks. Along with CPAL simulation capability (faster than real-time), we can easily write tasks that check that a property holds during the simulation of a certain scenario of execution, or during the execution of the real system. In addition, knowing that we will execute requirements force to write SMART requirements. For instance, requirement R4 for the smart parachute could be verified with the code shown in Figure 3. It is a classical implementation of an observer automaton for temporal logic properties. The designer can write tasks to stimulate the core design along with their observers to check the fulfillment of the requirements. Both design and validation of the software will be further explained in the following sections.


OK
on (pilotHasPressedTheButton)
EmergencyRequired
after (1430ms) if (not parachuteDeployed)
Fail

```
processdef R4Observer (
  in bool : pilotHasPressedTheButton,
  in bool : parachuteDeployed)
{
  state OK {
  }
  on (pilotHasPressedTheButton)
    to EmergencyRequired;
  state EmergencyRequired {
  }
  after (1430ms) if (not parachuteDeployed)
    to Fail;
  state Fail {
    /* println("R4 FAILED"); */
    assert(false);
  }
}
```

Figure 3: Formalisation of requirement R4: code on the right and visual representation on the left.

To counteract the natural tendency of focusing on functional aspects only, CPAL provides means to define precisely the timing behaviour of the system. For instance, CPAL language enforce the organisation in tasks with well-defined activation properties (e.g., periodic tasks with offsets). Moreover the CPAL editor show the Gantt chart of the task activations. This helps the designer verify that the timing behavior does not jeopardize the functional correctness of the system (e.g: excessive jitters, deadline misses, data are produced after being used, etc). This possibilities can be leveraged to verify requirements involving timing properties.

3 Software design and implementation

The system is really made up of two parts. One is called the transmitter, the other the receiver as seen in Fig. 1. The transmitter will is to be used by the pilot in case of emergency. The receiver will be embedded in the drone, and inserted between the RC receiver and the autopilot, so has to be able to turn the motor off and release the parachute. The CPAL code of the smart parachute system is available at `http://www.designcps.com/wp-content/uploads/ertss2016.zip`.

3.1 Functional architecture on sender and receiver side

The software architecture on the sender and receiver sides are quite similar as shown in Fig. 4 and 5. Both have in common 3 tasks (rounded rectangles):

- a task to manage the current global mode (`tm_modeTask` and `rcp_modeTask`),
- another to manage the user interface, *i.e.* LEDs in this implementation (`tm_uiTask` and `rcp_uiTask`),
- and a task dedicated to the wireless communication (Xbee in the prototype) (`tm_xbeeTask` and `rcp_xbeeTask`).

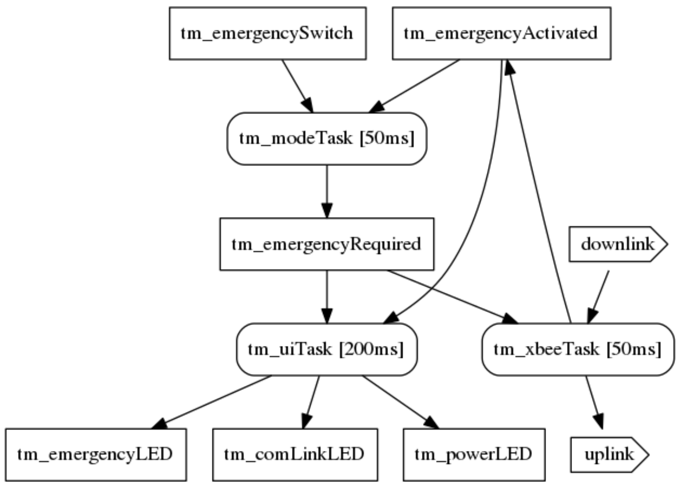


Figure 4: Software architecture of the transmitter: three tasks (rounded rectangles) `tm_modeTask`, `tm_uiTask`, and `tm_xbeeTask`, and their mean of communication (Rectangle and cds shape). `uplink` and `downlink` are channels of communication between transmitter and receiver, while the rectangles are global variables (screenshot from the CPAL editor).

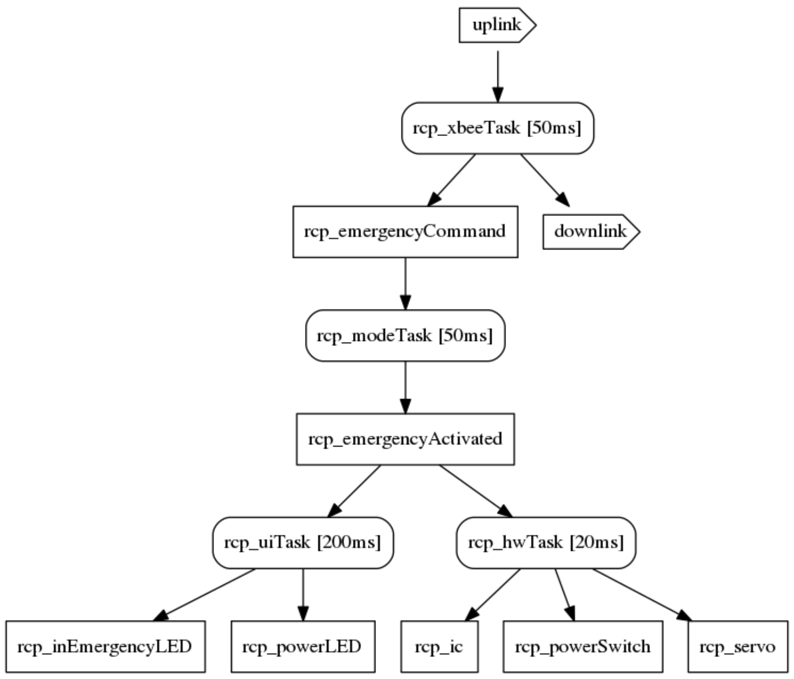


Figure 5: Software architecture of the receiver: Same architecture as the transmitter except one specific task `rcp_hwTask` to handle specifically the servo and electronic switches.

All rectangles in the Figures 4 and 5 are kinds of global variables used to share state information between tasks, referred to as *processes* in CPAL. The two others, `uplink` and `downlink,` are FIFO queues. In simulation, they are abstract Xbee messages while in the prototype implementation they are queues of characters as both Xbee modules are configured in transparent mode (*i.e.*, they implement a serial connection over-the-air, just like a wire would do). In the final prototype, they should be configured as Xbee frames so as to benefit from services of the Xbee protocol (e.g., “keep-alive” messages).

The receiving module executes another process, named `rcp_hawTask`, to control the ESCs (Electronic Speed Control), that reduces the motor's speed through a PWM signal command, and to trigger the opening parachute. Indeed requirement R3 requires to cut-off the motor before releasing the parachute. To do so, one need to by-pass the autopilot PWM commands with the `rcp_ic` electronic switch, send a zero command to the ESC via the `rcp_powerSwitch`, and finally release the parachute by setting the PWM `rcp_servo`.

3.2 Parachute deployment sequence

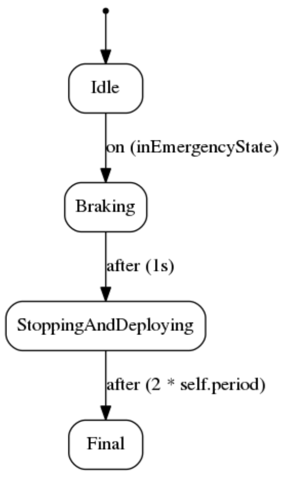


Figure 6: Logical sequence before parachute deployment.

The parachute deployment sequence, or any sequence with states and transitions between states, is easily described in CPAL. The CPAL editor also automatically displays the corresponding automaton graphically as shown on Figure 6. As seen on the figure, we let the motors brakes for 1s in order to delay the parachute deployment so that its strings do not get entangled in the moving propellers.

CPAL is not only a language but an execution platform. Indeed, with the same source code, one can interact physically with the GPIOs of a Raspberry Pi or a Freescale FRDM-K64F board, or with device drivers on an embedded Linux. Programming a GPIO, a PWM command, an analog-to-digital converter, etc, is simplified for the programmer since the low-level interactions with the hardware are performed by the interpreter. At run-time, the CPAL model is interpreted by an execution engine which relies on principles of similar systems deployed in interlocking systems, for instance at SNCF [7], where the hardware interpreter guarantees the semantics of the execution.

In terms of scheduling, the processes in charge of the user interface (`tm_uiTask` and `rcp_uiTask`) can be run at slower rate since they are meant to inform the user, for which a period of 200ms is sufficient. The communication tasks have a larger execution times than the other tasks because they are interacting with the Xbee module on the serial bus. It is crucial nonetheless that these communication tasks are able to react quickly, they are thus assigned an execution period of 50ms. The mode management task are on both data-flow, and, for this reason, is given a period of 50ms too.

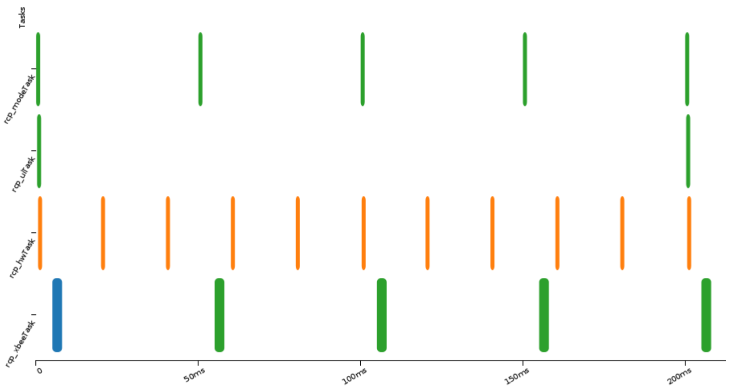


Figure 7: Gantt chart of the tasks activations on the receiver, from top to bottom: `rcp_modeTask`, `rcp_uiTask`, `rcp_hwTask` and `rcp_xbeeTask`. The color of a bar indicates the current state the task is in and the width of a bar indicates the task execution time.

The CPAL editor provides a timing diagram of the activation of all tasks, as shown on Figure 7 for the receiver. Design decisions can be taken by analyzing the timing behavior of the tasks on this Gantt diagram. For instance, in order to make the Xbee task really periodic, *i.e.* without any jitter, we have set an initial offset of 5ms for this task which will then not be delayed anymore by the other tasks. On the other hand, one observes that the `rcp_hwTask` is regularly slightly delayed by the `rcp_modeTask` and `rcp_uiTask`.

4 Verification of system correctness

4.1 Verification in nominal mode

The same CPAL source code can be run in real-time mode, where the execution of the code follows the timing specifications, typically tasks' periods, and it can also be run as fast as possible, in simulation mode. This latter mode enables to explore more trajectories of the system. Obviously such a verification by simulation will not be exhaustive, yet it is powerful in our experience (see [11]) and enforces good practices like formalizing requirements. For instance, we have included to the code on the receiving side the specification of requirement R4 and have checked by simulation of the model that the requirement holds.

```
@cpal:time {
  rcp_modeTask.wcet = 300us;
  rcp_uiTask.wcet = 300us;
  rcp_xbeeTask.wcet = 500us;
  rcp_hwTask.wcet = 500us;

  if (true) {
    rcp_modeTask.execution_time = rcp_modeTask.wcet;
    rcp_uiTask.execution_time = rcp_uiTask.wcet;
    rcp_xbeeTask.execution_time = rcp_xbeeTask.wcet;
    rcp_hwTask.execution_time = rcp_hwTask.wcet;
  } else {
    rcp_modeTask.execution_time = time64.rand_uniform( 2 * rcp_modeTask.wcet / 3, rcp_modeTask.wcet);
    rcp_uiTask.execution_time = time64.rand_uniform( 2 * rcp_uiTask.wcet / 3, rcp_uiTask.wcet);
    rcp_xbeeTask.execution_time = time64.rand_uniform( 2 * rcp_xbeeTask.wcet / 3, rcp_xbeeTask.wcet);
    rcp_hwTask.execution_time = time64.rand_uniform(2 * rcp_hwTask.wcet / 3, rcp_hwTask.wcet);
  }
}
```

Figure 8: Execution time annotations of tasks. The `execution_time` attribute of a task is used in simulation. Changing `true` to `false` in the code would simulate an execution time obeying a uniform distribution in the interval 2/3 of the WCET to the WCET, instead of always simulating the WCET.

CPAL offers support to express timing properties, which are often equally important as the functional behavior in real-time systems. For instance, it is possible to add timing annotations to a program, as an example is shown in Figure 8. With such annotations, typically obtained by on-target measurements, it is possible to simulate the effects of timing behaviors, typically the execution times of the tasks, or to perform a worst-case schedulability analysis (see [10] for the schedulability analysis of CPAL tasks).

In our context, different timing behaviors can change the worst-case delay until the complete deployment of the parachute, *i.e.* the delay between when the user pushes the red button and when the parachute is fully deployed. This delay does not only depend on the opening sequence, but also scheduling of the tasks, and the time needed to transmit the message. All this can be simulated in CPAL on the basis of the same functional model.

4.2 Model-based fault-injection

It is needed during the design of many systems with dependability constraints to study their resilience to errors and faults that can happen at run-time. A powerful technique for this purpose is fault-injection which can be done on models or on the actual system. Here, we are interested in studying the impact of message losses, which are likely to happen with wireless communication and thus pose a threat to the correct functioning of the smart parachute. We can for instance introduce between the receiver and transmitter tasks a CPAL process simulating a faulty network, or simulate the fact that the first emergency message is loss. What we have done here, is simulate the random loss of up-link messages depending on a network quality ratio, and analyze the effect on the fulfillment of requirement R4 about the latency in deploying of the parachute.

We conducted simulations with a network quality ratio varying from 50% to 100% by step of 10% (5000 simulations per step). A network quality ratio of 50% means here that on average 50% of the messages are successfully transmitted. For each simulation, we have counted the number of times requirement R4 is satisfied. We have also recorded the minimum, average, and maximum time needed by the parachute to be deployed. All these data are presented in Figure 9.

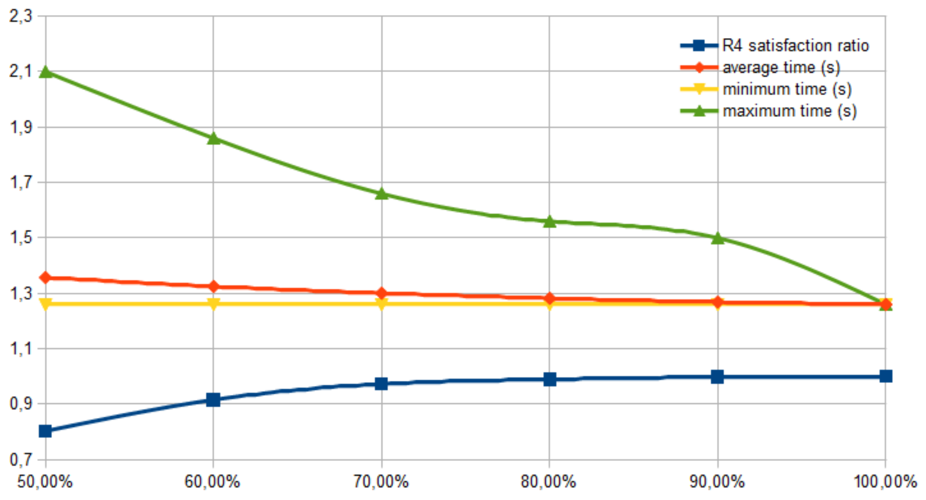


Figure 9: Time for the parachute to deploy (in seconds) and satisfaction of requirement R4 versus network quality ratio.

What we observe for instance, is that even with a 60% up-link network quality, R4 is satisfied 90% of the time. But the maximum deployment time can be then nearly half-more. It means that the parachute will have less time to decrease the speed of the UAV, and that the UAV will fall with greater energy on the ground. Obviously, the minimum time is constant and correspond to the critical path of the data-flow. The average time is not that different because once the emergency has been required, the transmitter kept sending an emergency command until it receives the acknowledgment that the sequence of deployment has been be triggered. Yet, without simulation, it would be difficult to derive the worst-case situation.

5 Conclusion

The paper reports on the development of a safe termination add-on component for UAS. Such a safety component improves the safety in the usual situation where the autopilot cannot be fully trusted. The component is developed in the CPAL language which has been designed to provide the right language abstractions to develop such embedded systems with dependability constraints. The use of model-interpretation makes it easier to verify the correctness of the system since the logic of the application is decoupled from the run-time services and written in a high-level language. The ability to easily express requirements in CPAL and verify them in simulation mode or real-time execution mode is also a powerful technique in our opinion.

The list of requirements of the termination system can be further refined, and the models extended accordingly. This ongoing work is being conducted on the basis of the experience gained on a prototype we are developing. The verification of the system correctness requires further work too, a question that needs to be investigated is whether the use of simulation alone can provide the verification coverage needed by such systems.

References

1. Ciarletta L., Guénard A. , “The AETOURNOS project: Using a flock of UAVs as a Cyber Physical System and platform for application-driven research”, EmSens 2012.
2. Ciarletta L., Guénard A., Presse Y., Galtier V., Song Y.Q, Ponsard J.C., Abarkane S., Theilliol D., ”Simulation and Platform Tools to develop safe flock of UAVs: a CPS Application-Driven Research”. In International Conference on Unmanned Aircraft Systems, ICUAS 2014.
3. UAV Challenge, “Search and Rescue Challenge - Mission, rules and judging criteria”, version 1.4, August 2014.
4. Public site of the ISO Working Group mandated to develop ISO/IEC 29110, `http://profs.etsmtl.ca/claporte/English/VSE/index.html`, retrieved 2015/10/21.
5. RTaW-ReqLab requirements editor, `https://www.requirements.fr`, 2015.
6. Van Lamsweerde A., “Requirements Engineering - From System Goals to UML Models to Software Specifications”, ISBN 0470012706, 2009.
7. Antoni M., “Formal validation method and tools for computerized interlocking system”, FM 2012, Industry day, slides available at `http://fm2012.cnam.fr/fm2012/ID2012-Marc-Antoni.pdf`.
8. Maraninchi F., Rémond Y., “Mode-Automata: a new Domain-Specific Construct for the Development of Safe Critical Systems”, Science of Computer Programming, Elsevier, n˚46, pp. 219-254, 2003.
9. Navet N., Fejoz L., Havet L., Altmeyer S, “Lean Model-Driven Development through Model-Interpretation: the CPAL design flow”, to appear at ERTS2016. Preliminary version available as technical report from the University of Luxembourg at `http://hdl.handle.net/10993/22279`.
10. Altmeyer S., Navet N., “The case for FIFO scheduling”, technical report from the University of Luxembourg, to appear, 2015.
11. Altmeyer S., Navet N., Fejoz L., “Using CPAL to model and validate the timing behaviour of embedded systems”, WATERS Workshop, July 2015. Available at `http://hdl.handle.net/10993/21250`.
12. “Arrêté du 11 avril 2012 relatif à la conception des aéronefs civils”, Annexe2-2.2.6, 2012. Available at `http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000025834953&dateTexte=20151022`
13. Clothiera R.,Williams B., Fulton N., “Structuring the safety case for unmanned aircraft system operations in non-segregated airspace”, Safety Science, vol.79, pages 213–228, November 2015.
14. “Sogilis, le projet CAP 2018 retenu au FUI20”, `http://www.aerospace-cluster.fr/news-entreprises/sogilis-le-projet-cap-2018-retenur-au-fui20/`, Retrieved November 13, 2015.

Towards Resilient Computing on ROS for Embedded Applications

Jean-Charles Fabre[1], Michal Lauer[2], Matthieu Roy
CNRS-LAAS, Ave du Colonel Roche,
F-31400 Toulouse, France
[1]Univ de Toulouse, INP, LAAS, F-31400 Toulouse, France

Matthieu Amy[1], William Excoffon[1], Miruna Stoicescu[3]
CNRS-LAAS, Ave du Colonel Roche, F-31400
Toulouse, France
[2]Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France
[3] Presently with ESOC/ESA, Darmstadt, Germany, on behalf of GMV

***Abstract*—Systems are expected to evolve during their service life in order to cope with changes of various natures, ranging from fluctuations in available resources to additional features requested by users. For dependable embedded systems, the challenge is even greater, as evolution must not impair dependability attributes. Resilient computing implies maintaining dependability properties when facing changes. Resilience encompasses several aspects, among which evolvability, i.e., the capacity of a system to evolve during its service life. In this paper, we discuss the evolution of systems with respect to their dependability mechanisms, and show how such mechanisms can evolve accordingly. From a component-based approach that enables to clarify the concepts, the process and the techniques to be used to address resilient computing, in particular regarding the adaptation of fault tolerance (or safety) mechanisms, we show how Adaptive Fault Tolerance (AFT) can be implemented with ROS. Beyond some implementation details given in the paper, we draw the lessons learned from this work and discus the limits of this runtime support to implement such resilient computing features in embedded systems.**

I. INTRODUCTION

Evolution during service life is inevitable in many systems today. A system that remains dependable when facing changes (new threats, change in failures modes, updates of applications) is called resilient. The persistence of dependability when facing changes is called resilience [1]. *Resilient computing* encompasses several aspects, among which evolvability, i.e., the capacity of a system to evolve during its service life. On the other hand, dependability relies on fault-tolerant computing at runtime, enabled by Fault Tolerance Mechanisms (FTMs) attached to the application. As such, one of the key challenges of resilient computing is the capacity to adapt the FTMs attached to an application during its operational life.

One important aspect of a dependable system design is the definition of the fault model. This fault model considers both hardware and software faults may lead to failure modes that impair the correct behavior of the system. In critical systems, such failure modes may violate safety properties. The role of the safety analysis (e.g. using the FMECA method, FMECA stands for *Failure Modes, Effects and Criticality Analysis*) is to identify the failure mode and then define the safety mechanisms to prevent the violation of safety properties. Such safety mechanisms rely on basic error detection and recovery mechanisms, namely fault tolerance techniques following Laprie's terminology. Such safety mechanisms are based on *Fault Tolerance Design Patterns* that can be combined together. The safety analysis is often done a priori according to the fault model that had been defined.

During the operational life of the system, several situation may occur. New threats may lead to revise the fault model (electromagnetic perturbations, obsolescence of HW components, Software aging, etc.). A revision of the fault model has consequences on the fault tolerance mechanisms to be used. In other words, the validity of the fault tolerance mechanisms of safety mechanisms (whatever you want to call them) depends on the representativeness of the fault model. In a certain sense, a bad choice of the fault model may lead to pay for useless mechanisms in both normal operation and erroneous situations. This has an obvious side effect on the performance and on the dependability measures (reliability, dependability) respectively. This means that a change in the definition of the fault model implies a change in the fault tolerance mechanisms.

Beyond the fault model, there are other sources of changes.

Resources changes may also impair some safety mechanisms that rely on hardware resources. A typical example is the lost of processing units, but simply a loss in networks bandwidth may invalidate some fault tolerance mechanisms from a timing viewpoint.

Application changes are more and more frequent during the operational lifetime. This is obvious for many conventional applications (e.g. mobile phones) but it is becoming also needed for more critical embedded systems. This is the case for long living systems like space or avionics systems, but also in the automotive domain, not only for maintenance purposes but also of commercial reasons. The evolution of the specification during the lifetime of a system is a fact, it follows the evolution of the user requirements or needs. The notion of versioning (updates) or the loading of additional features (upgrades) may lead to change the assumptions on top of which the implementation of FT mechanisms rely. Such change implies revisiting the FMECA spreadsheets but also the implementation of the FT mechanisms. Some FT mechanisms rely on strong assumptions regarding the behavior of the application, and everybody knows in the dependability community the importance of the coverage of such assumptions [16].

As a conclusion, the safety mechanism must remain compliant with all assumptions in terms of fault model, resources and application characteristics during the whole lifetime of the system. Their efficiency relies on this statement.

In this paper, we first motivate the issue and then report on an approach taking advantage of Component Based Software Engineering technologies for tackling this crucial aspect of resilient computing, namely the adaptation of fault tolerance mechanisms. We defined a minimal runtime support for implementing adaptive fault tolerance. The second part of this paper shows how this minimal runtime support can be implemented on ROS (*Robot Operating System*), presently used in many applications (robotics applications, automotive applications like ADAS – *Advanced Driver Assistance Systems*, or military applications). We illustrate the mapping of ideal components to ROS components and give implementation details of a fault tolerance design pattern that is adaptive at runtime. We finally draw the lessons learnt from our first experiments, discuss the limits of the exercise, and identify some promising directions.

In Section II we present the problem statement, and then summarize our *Component-Based Software Engineering* (CBSE) approach for adaptive fault tolerance in Section III. A full account of this approach can be found in [13]. The mapping of this approach to ROS is described in Section IV. The lessons learnt are given in Section V before concluding.

II. PROBLEM STATEMENT

The need for *Adaptive Fault Tolerance* (AFT) rising from the dynamically changing fault tolerance requirements and from the inefficiency of allocating a fixed amount of resources to FTMs throughout the service life of a system was stated in [2]. AFT is gaining more importance with the increasing concern for lowering the amount of energy consumed by cyber-physical systems and the amount of heat they generate [3]. For Dependable systems that cannot be stopped for performing off-line adaptation, on-line adaptation of *Fault Tolerance Mechanisms* (FTMs) has attracted research efforts for some time now. However, most of the solutions [4], [5], [6] tackle adaptation in a preprogrammed manner: all FTMs necessary during the service life of the system must be known and deployed from the beginning and adaptation consists in choosing the appropriate execution branch or tuning some parameters, e.g., the number of replicas or the interval between state checkpoints. Nevertheless, predicting all events and threats that a system may encounter throughout its service life and making provisions for them is impossible. The use of FTMs in real operational conditions may lead to slight updates or unanticipated upgrades, e.g., compositions of FTMs that can tolerate a more complex fault model than initially expected.

In both aeronautical and automotive systems, the ability to perform remote changes for different purposes is essential: maintenance but also updates and upgrades of embedded applications. The remote changes should be partial as it is unrealistic to reload completely an processing unit from small updates. This idea is recently promoted by some car manufacturers like Renault, BMW but also TESLA Motors in the USA stating in its website *"Model S regularly receives over-the-air software updates that add new features and functionality"*. It is important to mention that performing remote changes will become very important for economic reasons, for instance selling options a posteriori since most of the evolution in the next future will rely on software for the same hardware configuration (sensors and actuators). In addition to this, the X-to-X applications (X being cars, planes or any smart critical objet) will imply rapid adaptation of onboard software to remain consistent with the network of X.

We propose an alternative to preprogrammed adaptation that we denote *agile adaptation of FTMs*. The term "agile" is inspired from agile software development [7] that emphasizes the importance of accommodating change during the lifecycle of an application at a reasonable cost, rather than striving to anticipate an exhaustive set of requirements. Agile adaptation of FTMs enables systematic evolution: according to runtime observations of the system and of its environment, new FTMs can be designed off-line and integrated on-line in a flexible manner, with limited impact on the existing software architecture.

Evolvability has long been a prerogative of the application business logic. A rich body of research exists in the field of software engineering consisting of concepts, tools, methodologies and best practices for designing and developing adaptive software [8]. Consequently, our approach for the agile adaptation of FTMs leverages advancements in this field such as Component-Based Software Engineering [9], Service Component Architecture [10] and Aspect-Oriented Programming [11].

The basic idea is the following. Fault Tolerance or Safety Mechanisms are developed as a composition of elementary mechanisms, e.g. basic design patterns for fault tolerance computing.

Using such concepts and technologies, **we design FTMs as "Lego"-like brick-based assemblies** that can be methodically modified at runtime through fine-grained changes affecting a limited number of bricks. This is the basic idea of our approach that maximizes reuse and flexibility, contrary to monolithic replacements of FTMs found in related work, e.g., [4], [5], [6].

However, most of software runtime supports used in embedded systems today do not rely on dynamic CBSE concepts. AUTOSAR, for instance, relies on very static system engineering concepts and does not provide today much flexibility [12]. A new approach enabling remote updates to be carried out, including for safety mechanisms, is required.

ROS seems an appealing candidate for the dynamic composition of safety mechanisms. ROS is described as[1]: *ROS is an open-source, meta- operating system for your robot*. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS can be viewed as a middleware running on top of a Unix-based operating system (typically Linux). ROS is used in robotics applications (e.g. Robonaut 2 from NASA within the ISS) but also in other industry sectors, the automotive industry for instance. This middleware provides a *weak* component

1 ttp://wiki.ros.org/ROS/Introduction

approach and means to dynamically manipulate system configuration. It is open-source, its user community is very large and it is used for critical application e.g. at NREC (The *National Robotics Engineering Center* in Pittsburgh) for unmanned military vehicles (e.g. the *Crusher*).

III. Adaptive Fault Tolerance

A. Basic concepts for AFT

Some basic concepts must be discussed to address the problem of Adaptive Fault Tolerant computing. Three essential concepts must be discussed beforehand:

- *Separation of concerns:* this concepts is now well known, it implies a clear separation between the functional code, i.e. the application, and the non-functional code, i.e. the fault tolerance mechanisms in our case. The connection between the application code and the FTM must be clearly defined as specific connections. This means that the FTMs can be disconnected and replaced by a new one provide the connectors remains the same.
- *Componentization*: this concepts means that any software components can be decomposed into smaller components. Each component exhibit interfaces (services provided) and receptacles (services required). This means that any FTMs can be decomposed into smaller pieces, and conversely that an FTM is the aggregation of smaller. The ability to manipulate the binding between components (off-line but also on-line) is of high interest for AFT.
- *Design for adaptation*: the adaptation of software systems imply that (i) the software itself has been analyzed with adaptation in mind for later evolution using componentization (although all situations cannot be anticipated) and (ii) designed to simplify their adaptation including from a programming viewpoint (e.g. using object-oriented, aspect-oriented programming concepts).

Such basic concepts have been established and validated through various steps of analysis of fault tolerance design patterns and after several design and implementation loops, as discussed in [17].

The main benefits of AFT with respect to pre-programmed adaptation is clear, it provides means to define and update dependability mechanisms later during the lifetime of the system. Pre-program adaptation implies that all possible undesirable situations are defined at design time, which is difficult to anticipate regarding new threats (attacks), new failure modes (obsolescence of components), or simply adverse situations that have been ignored or forgotten during the safety analysis. In short, fine grain adaptation of FTMs improves maintainability of the system from a non-functional viewpoint.

B. Change Model

The choice of an appropriate fault tolerance mechanism (FTM) for a given application depends on the values of several parameters. We consider three classes of parameters: 1) fault tolerance requirements (FT); 2) application characteristics (A); 3) available resources (R). We denote (FT,A,R) as *change model*. At any point in time, the FTM(s) attached to an application component must be consistent with the current values of (FT, A, R).

The three classes of parameters enable to discriminate FTMs. Among fault tolerance requirements FT, we focus, for the time being, on the fault model that must be tolerated. Our fault model classification is based on well-known types [14], e.g., crash faults, value faults, development faults. In this work, we focus on hardware faults but the approach is perfectly reproducible for FTMs that target development faults.

The application characteristics A that we identified as having an impact on the choice of an FTM is: application statefulness, state accessibility and determinism. We con- sider the FTMs are attached to a black-box application. This means there is no possibility to interfere with its internals, for tackling non-determinism, for instance, in case an FTM only works for deterministic applications. Resources R play an important part and represent the last step in the selection process. FTMs require resources such as bandwidth, CPU, battery life/energy. In case more than one solution exists, given the values of the parameters FT and A, the resource criterion can invalidate some of the solutions. A cost function can be associated to each solution, based on R.

Any parameter variation during the service life of the system may invalidate the initial FTM, thus requiring a transition towards a new one. Transitions may be triggered by new threats, resource loss or the introduction of a new application version that changes the initial application characteristics. A particularly interesting adaptation trigger is the fault model change. Incomplete or misunderstood initial fault tolerance requirements, environmental threats such as electromagnetic interferences or hardware aging may change the initial model to a more complex one.

C. FT Design Patterns and Assumptions

To illustrate our approach, we consider some fault tolerance design patterns (design patterns of FTMs) and discuss their underlying assumptions and resource needs. Any change that invalidates an assumption or implies an unacceptable resource change calls for an update of the FTMs.

Duplex protocols tolerate crash faults using passive (e.g. *Primary-Backup Replication* denoted PBR), or active replication strategies (e.g. *Leader-Follower Replication* denoted LFR). In this case, each replica is considered as a *self-checking* component, the error detection coverage is perfect. The fault model includes hardware faults or random operating system faults (no common mode faults). At least 2 independent processing units are necessary to run this FTM.

Two design patterns tolerating transient value faults are briefly discussed here. *Time Redundancy* (TR) tolerates transient physical faults or random runtime support faults using repetition of the computation and voting. This is way to improve the self-checking nature of a replica, but it introduces a timing overhead. *Assertion&Duplex* (A&D) tolerates both transient and permanent faults. It's a combination of a duplex strategy with the verification using assertions of safety properties that could be violated by a value fault or by a random runtime support error.

Assumptions / FTM		PBR	LFR	TR	A&D
Fault Model (FT)	crash	✓	✓		✓
	transient			✓	✓
Application behaviour (A)	Deterministic		✓	✓	(✓)
	State access	✓			(✓)
Resources (R)	Bandwidth	high	low	nil	(TDB)
	# CPU	2	2	1	2

Fig. 1. Assumptions and fault tolerance design patterns charateristics

The underlying characteristics of the considered FTMs, in terms of (FT,A,R), are shown in Fig. 1. For instance, PBR and LFR tolerate the same fault model, but have different A and R. PBR allows non-determinism of applications because only the Primary computes client requests while LFR only works for deterministic applications as both replicas compute all requests. LFR could tackle non-determinism if the application was not considered a black-box, as in our approach. PBR requires state access for checkpoints and higher network bandwidth (in general), while LFR does not require state access but generally incurs higher CPU costs (and, consequently, higher energy consumption) as both replicas perform all computations.

During the service life of the system, the values of the parameters enumerated in Fig. 1 can change. An application can become non-deterministic because a new version is installed. The fault model can become more complex, e.g., from crash-only it can become crash and value fault due to hardware aging or physical perturbations. Available resources can also vary, e.g., bandwidth drop or constraints in energy consumption. For instance, the PBR→LFR transition is triggered by a change in application characteristics (e.g. inability to access application state) or in resources (bandwidth drop), while the PBR→A&D transition is triggered by a change in the considered fault model (e.g. safety property verification). Transitions can occur in both directions, according to parameter variation.

The priority is the fault model, the selection of the solution (i.e. the composition of several FTMs) depending on the application characteristics and the available resources. The final objective is always to comply with the dependability properties during the service lifetime.

D. *Design for adaptation of FTMs*

Our *"design for adaptation"* aims at producing reusable elementary components that can be combined to implement a given fault tolerance or safety mechanism. Any FTM follows the generic *Before-Proceed-After* metamodel. Many FTMs can be mapped and combined using this model, as shown in Fig. 2.

FTM	Before	Proceed	After
PBR (primary)		Compute	Checkpointing
PBR(backup)			State update
LFR (leader)	Forward request	Compute	Notify
LFR (follower)	Handle request	Compute	Handle notification
TR	Save/restore state	Compute	Compare
A&D		Compute	Assert

Fig. 2. Generic execution scheme for FT design patterns

Composition implies nesting the *Before-Proceed-After* metamodel. This approach improves flexibility, reusability, composability and reduces development time. Updates are minimized since just few components have to be changed.

E. *Runtime support*

The software runtime support must provide key features to manipulation the component graph. Any application or an FTM is perceived as a graph of components. From previous experiments reported in [17], the following primitive are required.

- Dynamic creation, deletion of components;
- Suspension, activation of components;
- Control over interactions between components for the creation and the removal of connections (bindings);

Our first implementation was done on a reflective component-based middleware, FRASCATI [14] providing a scripting language to manipulate the component graph, FScript [15]. The proposed approach is reproducible on any other support that provides these features.

IV. Adaptive Fault Tolerance on ROS

The main goal of ROS is to allow the design of modular applications: a ROS application is a collection of programs, called nodes, interacting only through message passing. Developing an application involve the assembly of nodes, which is akin to component-based approaches. Such an assembly is referred to as the computation graph of the application.

A. *Component model and reconfiguration*

Two communication models are available in ROS: a publisher/subscriber model and a client/server one. The publisher/subscriber model defines one-way, many-to-many, asynchronous communications through the concept of topic. When a node publishes a message on a topic, it is delivered to every nodes subscribing to this topic. Note that a publisher is not aware of the subscriber to its topic nor the other publishers. The client/server model defines bidirectional transaction (one request/one reply) synchronous communications through the concept of service. A node providing a service is not aware of the client nodes that may use its service. These high-level communication models allows to add, replace or delete nodes in a transparent manner, either offline or online.

To provide this level of abstraction, each ROS application includes a special node called the ROS Master. It provides registration and lookup services to the other nodes. All nodes register services and topics to the ROS master. It is the only node that has a comprehensive view of the computation graph. When a node issues a service call, it queries the master for the address of the node providing the service and then it sends its request to this address.

In order to be able to add fault-tolerance mechanisms to an existing ROS application in the most transparent manner, we need to implement interceptors. An interceptor provides a means to insert functionality, such as safety or monitoring nodes, into the invocation path between two ROS nodes. To this end, a relevant ROS feature is its remapping capability. At launch time, it is possible to reconfigure the name of any services or topics used by a node. Thus, requests and replies between nodes can be rerouted to interceptor nodes.

B. Implementing a componentized FT design pattern

A full implementation on ROS of a duplex FT design pattern, a *Primary Backup Replication* (PBR) combined with a *Time-Redundancy* (TR) design pattern is developed here.

1) Generic Computation Graph

We have identified a generic pattern for the computation graph of a FTM. Figure 3 shows its application in the context of ROS. Node *Client* uses a service provided by *Server*. The FTM computation graph is inserted between the two thanks to the ROS remapping feature. Since *Client* and *Server* must be re-launched for the remapping to take effect, the insertion is done offline. The FTM nodes, topics, and services are generic for every FTM discussed in section II. Implementing a FTM consist in specializing the *before*, *proceed*, and *after* nodes with its corresponding behavior (see Fig. 3).

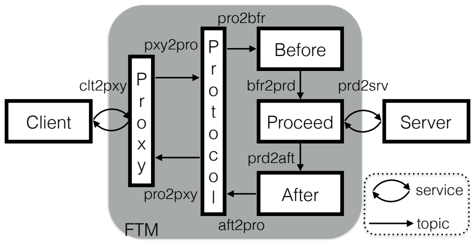


Fig. 3. Generic computation graph for FTM

We illustrate the approach, through a Primary-Backup Replication (PBR) mechanism added to the Client/Server application in order to tolerate a crash fault of the Server. Fig. 4 presents the associated architecture. Three machines are involved: the *Client*, which is also hosting the ROS, master, the *MASTER* site hosting the primary replica and the *SLAVE* site hosting the backup replica. For the sake of clarity, the symmetric topics and services between *MASTER* and *SLAVE* are not represented. Elements of the slave are suffixed with "_S".

2) Implementing PBR

We present the behavior of each node, the topics/services used through a request/reply exchange between a node *Client* and node *Server* (see Fig. 4).

- *Client* sends a request to *Proxy* (service *clt2pxy*);
- *Proxy* adds an identifier to the request and transfers it to *Protocol* (topics *pxy2pro*);
- *Protocol* checks whether it is a duplicate request: if so, it sends directly the stored reply to *Proxy* (topics *pro2pxy*). Otherwise, it sends the request to *Before* (service *pro2bfr*);
- *Before* transfers the request for processing to *Proceed* (topics *bfr2prd*); no action is associated in the PBR case, but for other duplex protocol, *Before* may synchronize with the other replicas;
- *Proceed* calls the actual service provided by *Server* (service *prd2srv*) and forwards the result to *After* (topics *prd2aft*);
- *After* gets the last result from *Proceed*, captures *Server* state by calling the state management service provided by the server (service *aft2srv*), and builds a checkpoint based on this information which it sends to node *After_S* of the other replica (topics *aft2aft_S*);
- *Protocol* gets the result (topics *aft2pro*) and sends it to *Proxy* (topics *pro2pxy*);
- On the backup replica, *After_S* transfers the last result to its protocol node *Proto_S* (topics *aft2pr_S*) and sets the state of its server to match the primary.

In parallel with request processing, the node crash detector on the *MASTER* (noted CD) periodically gives a proof of life to the crash detector (CD_S) on the *SLAVE* to assert its liveliness (topics *CD2CD_S*). If a crash is detected, then the crash detector of the slave notifies the recovery node (topics *CD_S2rcy*). This node has two purposes: (i) in order to enforce the fail-silent assumption, it must ensure that every node of the *MASTER* are removed; (ii) it switches the binding between the *Client* proxy and the *MASTER* protocol to the *SLAVE* protocol. Thus, the *SLAVE* will receive the *Client's* requests and will act as the *Primary*, changing its operating mode.

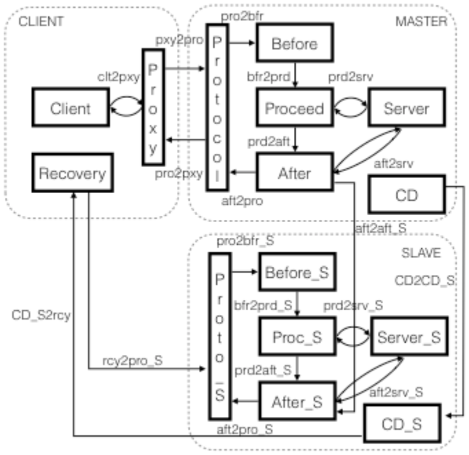


Fig. 4. Computation graph of a PBR mechanism

ROS does not provide APIs to dynamically change bindings between nodes. The node developer must implement the transition logics. The *SLAVE protocol* spins waiting for a notification from *recovery* (topics *rcy2pro_S*). This is done using the ROS API: background threads, within a node, check for messages independently of the node's main functionality. Upon reception of this topic, *protocol* subscribes to topic *pxy2pro* and publishes to topic *pro2pxy*. After this transition, the proxy forwards the *Client*'s requests to the *Slave* protocol.

3) Impact on the existing application

From the designer viewpoint, there are two changes required to integrate a FTM computation graph to its application. First, Client will have to be remapped offline to call the *proxy* node's service instead of directly the Server. Second, state management services, to get and set the state of the node, must be integrated to the *Server*. Form an object-oriented viewpoint any server inherits from an abstract class *stateManager* providing two virtual methods, *getState* and *setState*, overridden during the server development.

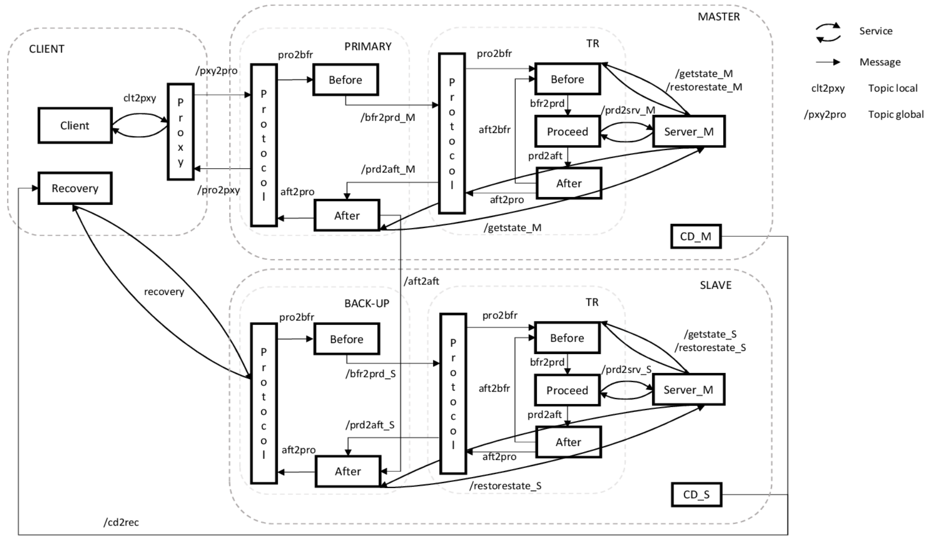


Fig. 5. Composition principle of FT mechanisms (PBR+TR).

C. Composition of FT mechanisms

The generic computation graph for FTM is designed for composability. In this section, the composition scenario is two-fold. We first illustrate the composition of two FTMs, PBR for crash faults and TR for transient value faults. Initially the application was installed with PBR. From an operational standpoint, at a given point in time, transient faults impacting numerical calculations appeared due to hardware components aging or sudden increase of environmental radiations. In a second step, later on, we consider that the communication channel between client and server can be the target for intrusions. Cryptographic protocols, based for instance on a simple *Public Key Infrastructure* (PKI), can be used to cipher communications and add cryptographic signatures.

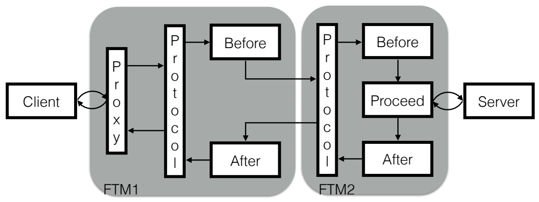


Fig. 6. Composition principle of FT mechanisms.

1) Composition of PBR and TR on ROS

With respect to request processing, a Protocol node and a Proceed node present the same interfaces: a request as input, a reply as output. Hence, a way to compose mechanisms is to replace the Proceed node of a mechanism by a Protocol and its associated Before/Proceed/After nodes, as shown in Fig. 6.

Our approach enables developing a new mechanism on the foundation of several existing ones. This improves the development time and the assurance in the overall system, since all mechanisms have been validated off-line by test and fault injection techniques.

The architecture of the composite FTM made of PBR and TR is given in Fig. 5. This figure is an extension of Fig. 4 where the *Proceed* node of the PBR has been replaced with the *Protocol* node of the TR implementation.

2) Composing FTM with Cryptographic protocols

The generic computation graph presented in Fig. 3 enables cryptographic protocols to be seamlessly added to an application, already equipped with accidental fault tolerance mechanisms, PBR and TR in our example. The cryptographic mechanism (called SEC for security) is located at both the client (SEC_C) and the server side (SEC_S) as shown in Fig. 7). On the server side, SEC operates before PBR and TR.

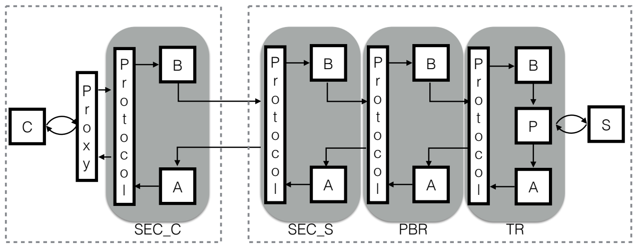


Fig. 7. Composition principle of SEC with other FT mechanisms.

In this example, we only deal with possible intrusions between the client and the server.

We assume that a node implements the *Certification Authority* (CA). Three topics are used to communicate with the CA, namely *Cli2CA* for the *Client*, *Master2CA* for the Master and *Slave2CA* for the Slave. The topic *Cli2CA* enables the *Before* node of the *Client* to collect the certificate of the Server. Similarly, the topic *Master2CA* and *Slave2CA* enable *Before* of the *Master*, respectively the *Slave*, to collect the certificate of the *Client*. We assume that all parties know CA's public key. We assume that, for each participant, *Client* or Server, *Before* and *After* of the SEC mechanism share the pair of private and public keys they received when initialized.

Before of the Client can then ciphers the request with K_{pub}^{S}, the *Server's public key,* and adds a signature, using K_{priv}^{C} the *Client's private key*;

Using the generic scheme given in Fig. 6, a message is sent by the client to the server side through a new topic (called *Client_2_Server*) connecting *Before* of SEC_C to *Protocol* of SEC_S.

Before of the Master deciphers the request with K_{priv}^{S}, *the Server's private key,* and checks the signature, using K_{pub}^{C}, the *Client's public key*;

The *Server* can then proceed with a valid deciphered request through PBR and TR.

Conversely, *After* of the *Master* ciphers the reply and computes a signature. *After* of the *Client* deciphers the reply, checks the signature, and finally delivers the reply to the *Client*.

The communication between *Master* and *Slave* can also be secured using a similar security protocol.

V. DYNAMIC COMPOSITION: TO WHAT EXTENT WITH ROS

A. Dynamic Adaptation of FTM

Dynamic adaptation of FTM is required to provide continuity of service in resilient systems. The question is then: is it possible to safely adapt a FTM at runtime in the context of ROS? A set of minimal API required to guarantee the consistency of the transition between two different FTMs has been established in previous work [14]:

- Control over components life cycle at runtime (add, remove, start, stop).
- Control over interactions between components at runtime, for creating or removing bindings.

Furthermore, ensuring consistency before, during and after reconfiguration, requires that no requests or replies are lost:

- Components are stopped in a quiescent state, i.e. when all internal processing has finished
- Incoming requests on stopped components must be buffered

With the exception of *add* and *remove*, ROS does not provide these APIs. However, these APIs can be emulated with dedicated logics in some nodes. For instance, we are using some binding control in the Primary to Backup switch described in our example. Controlling node lifecycle is more complex but can be done in the same manner and these principles can be applied in the context of dynamic adaptation, i.e. add new nodes at runtime and binding them in the computation graph.

The *protocol* node plays a central part to provide proper consistency during a transition. Indeed, our design pattern for FTM is such that only stateless nodes, namely *before*, *proceed* and *after,* need to change in order to switch from one FTM to the next. Thus, *protocol* does not need to be changed during a transition and it can be used to buffer messages and detect when the changing nodes are in quiescent state. To do this, *protocol* is extended to deal with three new messages. The first one is used to signal *protocol* that a transition is about to happen and it has to start storing incoming requests. The second one is published by *protocol* and confirms that the FTM is in a safe state and transition can be safely executed. In particular, the safe state is reached when *protocol* has received the replies of all pending requests. The third message is used to signal *protocol* that the transition has been executed and it can resume normal operation and release the requests stored during the transition.

Note that the described transition technique requires that an FTM is already in place in the system, meaning that the Client and the Server are already configured to use our proxy nodes. Installing an FTM in an application without interruption is not possible with ROS since control over binding at runtime is only possible with dedicated code within the nodes.

B. Implementing Dynamic Binding on ROS

Dynamic binding is not a core feature of ROS. As far as AFT is concerned, this is a major concept for runtime adaptation. However, *ROS* does not contain any API to control bindings online. In *ROS*, connections between nodes are based on pre-defined *Topics* and messages are sent/received through ports.

A *Topic* is defined by:

- A *name*: ports are connected through a named *Topic*.
- A *sending port*: *Publisher* or *Client* sends messages.
- A *receiving port*: *Subscriber* or *Server* receives messages.
- A *data type*: a *Topic* is assigned a data type for messages.

Several *Publishers* and *Subscribers* can communicate on the same *Topic* according to a unique message format, a given data type. The connection of a new node to the system implies creating a new *Topic* with its own data type. Suppose that *Node* A and B are connected to a *Node* C. When the data type from A to C and B to C is different, then two *Topics* are needed. If the same data type is used, then just one *Topic* is needed.

We defined two types of dynamic bindings: a) dynamic binding on *Pre-Defined Topics (PDT)*; b) dynamic binding on *UnAnticipated Topics (UAT)*.

Some topics can be pre-defined, for instance two topics, one between the Client and the primary, one between the Client and the backup in a PBR replication strategy. Others topics are unanticipated: some new topics are needed when a new node is created with a new data type for messages. This might be needed for the on-line composition of FTM later during the lifetime of the system.

<u>*Dynamic binding on PDT:*</u> This is the simpler case since *Topics* preexist in the ROS configuration. For example, in the PBR replication strategy, the two *Protocols* nodes (in the two replicas) are bound to the same topic, but the *Slave*'s port is deactivated. The *Proxy* sends the request but only one *Protocol* node receives it.

A third *Node*, in our implementation the *Recovery* node, is used to activate the *Slave*'s port when the *Master* crashes. A dedicated topic is defined and used to this aim. After recovery, the former *Slave*, i.e. new *Master*, can now listen to the *Proxy* and receive messages. It is the simplest way to dynamically bind *Nodes* since the same data type is used in this case.

<u>*Dynamic binding on UAT:*</u> In the case of unanticipated topics, the binding is a bit more difficult to achieve. Instead of reactivating a port, two communication ports must be created. Suppose that two nodes A and B must be connected at a given point in time through a new topic. The solution is based on:

- two methods added to both A and B to create ports, one for the publisher, one for the subscriber;
- a third node used to trigger and control the creation of the channel (activation of the methods).

The *Topic* defined offline corresponds to one data type that is handled by the methods. The third *Node* is part of the implementation of AFT, in fact part of the implementation of an adaptation engine responsible for the manipulation the FTM configuration.

VI. Lessons learnt and conclusion

Installing an FTM within a ROS application or adapting an existing FTM does not incur technical difficulties as long as the system's nodes (application + FTM) can be stopped and re-launched. Indeed, using the remapping capability of ROS implies rewriting some configuration files, which are taken into account only during the initialization of the nodes. For system where interruption of service is not an option, adaptation has to be done at runtime. In the context of ROS, this requires some additional software development.

Regarding the features of ROS for implementing AFT, we can say that they are not fully satisfactory. The main troubles relate to the dynamic binding on unanticipated topics and on the weak API to control components at runtime. However, ROS provides separation of concerns, since component can be mapped to nodes (Unix processes) that have their own address space. Dynamic binding is possible on pre-defined topics. For unanticipated topics, a customized solution was proposed in this work. Control over components relies on the underlying operating system to suspend and activate nodes, i.e. processes and threads, and to store input messages. However, ROS is an acceptable candidate for AFT, in other words, resilient computing using AFT can be implemented on ROS.

Regarding safety issues, the design of AFT and its validation is always carried out off-line. Any composition of mechanisms due to a change in the various axis of the change model denoted (FT, A, R) follows a design and validation process off-line that can be conformant to standards like DO178C or ISO26262, to comply with certification if needed.

Some performance measurements have been obtained. The overhead of the FTM (composition of PBR+TR) is less than 10 ms (on a PC, Intel I7 Quad Core, 8 Go RAM). Actually, the real overhead is very dependent of the complexity of the application, in particular the handling of the application state, and the network performance. As a conclusion, the implementation of AFT on ROS is independent from the application and the network.

References

[1] J.-C. Laprie, “From Dependability to Resilience,” in 38th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2008.

[2] K. H. K. Kim and T. F. Lawrence, “Adaptive Fault Tolerance: Issues and Approaches,” in Proceedings of the Second IEEE Workshop on Future Trends of Distributed Computing Systems. IEEE, 1990, pp. 38–46.

[3] C. Krishna and I. Koren, “Adaptive Fault-Tolerance for Cyber- Physical Systems,” in IEEE International Conference on Computing, Networking and Communications (ICNC), 2013, pp. 310–314.

[4] J. Fraga, F. Siqueira, and F. Favarim, “An Adaptive Fault- Tolerant Component Model,” in 9th Workshop on Object- Oriented Real-Time Dependable Systems. IEEE, 2003, pp. 179–186.

[5] L. C. Lung, F. Favarim, G. T. Santos, and M. Correia, “An Infrastructure for Adaptive Fault Tolerance on FT-CORBA,” in 9th International Symposium on Object and Component- Oriented Real-Time Distributed Computing. IEEE, 2006.

[6] O. Marin, P. Sens, J.-P. Briot, and Z. Guessoum, “Towards Adaptive Fault-Tolerance for Distributed Multi-Agent Sys- tems,” in 4th European Research Seminar on Advances in Distributed Systems, 2001, pp. 195–201.

[7] J.HighsmithandA.Cockburn,“AgileSoftwareDevelopment: The Business of Innovation,” *Computer*, vol. 34, no. 9, pp. 120–127, 2001.

[8] P. McKinley, S. Sadjadi, E. Kasten, and B. H. C. Cheng, “Composing Adaptive Software,” *Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[9] [11] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[10] [12] J. Marino and M. Rowley, *Understanding SCA (Service Component Architecture)*. Addison-Wesley Professional, 2009.

[11] [13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-oriented program- ming,” *ECOOP'97Object-Oriented Programming*, pp. 220– 242, 1997.

[12] H. Martorell, J.-C. Fabre, M. Lauer, M. Roy and R. Valentin. Partial Updates of AUTOSAR Embedded Applications — To What Extent?, in European Dependable Computing Conference (EDCC), 2015, Paris, France.

[13] M.Stoicescu, J.-C. Fabre, M. Roy, From Design for Adaptation to Component-Based Resilient Computing. PRDC 2012: 1-10

[14] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schi- avoni, and J.-B. Stefani, “A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures,” *Software: Practice and Experience*, 2011.

[15] M. Leger, T. Ledoux, and T. Coupaye, “Reliable Dynamic Reconfigurations in a Reflective Component Model,” *13th International Conference on Component-Based Software En- gineering*, 2010.

[16] D. Powell, "Failure Mode Assumption and Assumpion Coverage", in Proc. of the IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-22), Boston (USA), 1992, pp.386-395. (revised in the book Predictably Dependable Computing Systems, ISBN 3-540-59334, 1995.)

[17] M.Stoicescu, "Architecting Resilient Computing Systems: A Component-based Approach", PhD thesis, National Polytechnic Institute of Toulouse (INP), 2013. ww.theses.fr/en/2013INPT0120.

Session 13

Code Generation

Thursday 28th, 11:45 – Auditorium St Exupery

RTE Generation and BSW Configuration Tool-Extension for Embedded Automotive Systems

Georg Macher*‖,Rene Obendrauf‖, Eric Armengaud‖, Eugen Brenner* and Christian Kreiner*

*Institute for Technical Informatics, Graz University of Technology, AUSTRIA
Email: {georg.macher, brenner, christian.kreiner}@tugraz.at

‖AVL List GmbH, Graz, AUSTRIA
Email: {georg.macher, rene.obendrauf, eric.armengaud}@avl.com

***Abstract*—Automotive embedded systems have become very complex, are strongly integrated and the safety-criticality and real-time constraints of these systems are raising new challenges. Distributed system development, short time-to-market intervals, and automotive safety standards (such as ISO 26262 [8]) require efficient and consistent product development along the entire development lifecycle. The challenge, however, is to ensure consistency of the concept constraints and configurations along the entire product life cycle. So far, existing solutions are still frequently insufficient when transforming system models with a higher level of abstraction to more concrete engineering models (such as software engineering models).**

The aim of this work is to present a model-driven system-engineering framework addon, which enables the configurations of basic software components and the generation of a runtime environment layer (RTE; interface between application and basic software) for embedded automotive system, consistent with preexisting constraints and system descriptions. With this aim in mind a tool bridge to seamlessly transfer artifacts from system development level to software development level is described. This enables the seamless description of automotive software and software module configurations, from system level requirements to software implementation and therefore ensures both consistency and correctness for the configuration.

***Keywords*—automotive, embedded systems, Model-based development, basic software configuration, traceability, model-based software engineering.**

I. Introduction

Embedded systems are already integrated into our everyday lives and play a central role in all domains including automotive, aerospace, healthcare, manufacturing industry, energy, or consumer electronics. Current premium cars implement more than 90 electronic control units (ECU) per car with close to 1 Gigabyte software code [4], these are responsible for 25% of vehicle costs and bring an added value between 40% to 75% [18]. This trend of making use of modern embedded systems, which implement increasingly complex software functions instead of traditional mechanical systems is unbroken in the automotive domain. Similarly, the need is growing for more sophisticated software tools, which support these system and software development processes in a holistic manner. As a consequence, the handling of upcoming issues with modern real-time systems, also in relation to ISO 26262 [8], model-based development (MBD) would appear to be the best approach for supporting the description of the system under development in a more structured manner. Model-based development approaches enable different views for different stakeholders, different levels of abstraction, and provide a central storage of information. This improves the consistency, correctness, and completeness of the system specification. Nevertheless, such seamless integrations of model-based development are still the exception rather than the rule and frequently MBD approaches fall short due to the lack of integration of conceptual and tooling levels [3].

The aim of this paper is to present a tool approach which enables a seamless description of safety-critical software, from requirements at the system level down to software component implementation in a bidirectional way. With the presented tool available hardware- software interfacing (HSI) information can be used to generate basic software (BSW) component configurations, as well as, automatic generation of the runtime environment layer (RTE; interface between application software (ASW) and basic software).

The tool consists of a basic software configuration generator and a software interface generator producing $.c$ and $.h$ files for linking ASW and BSW. To ensure more versatility of the tool the required HSI information can either be imported from a HSI spreadsheet template or the system model representation. The goal is, on one hand, to support a consistent and traceable refinement from the early concept phase to software implementation, and on the other hand, to combine the versatility and intuitiveness of spreadsheet tools (such as Excel) and the properties of MDB tools (e.g., different views, levels of abstraction, central source of information, and information reuse) bidirectionally to support semi-automatic generation of BSW configuration and the SW-SW interface layer (in AUTOSAR notation known as runtime environment - RTE).

The document is organized as follows:
Section II presents an overview of related works as well as the fundamental model-based development tool chain on which the approach is based. In Section III a description of the proposed tool and a detailed depiction of the contribution parts is provided. An application and evaluation of the approach is presented in Section IV. Finally, this work is concluded in Section V with an overview of the presented approach.

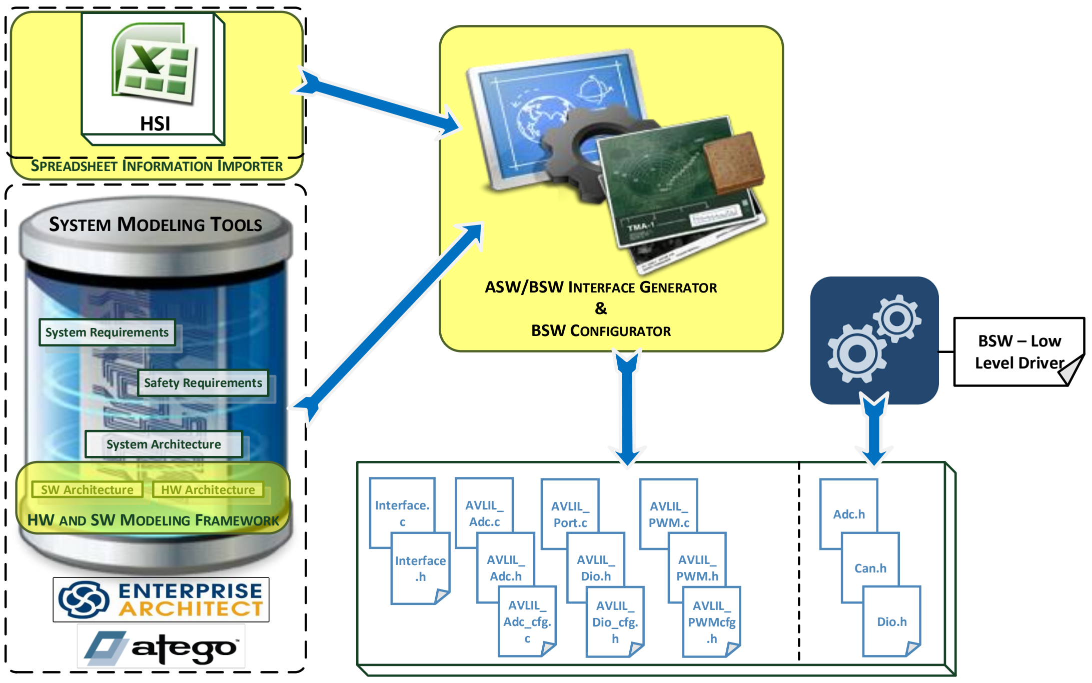


Fig. 2. Portrayal of the Approach for Generation of BSW Configuration and SW Interfaceing Files for the SW Development Phase

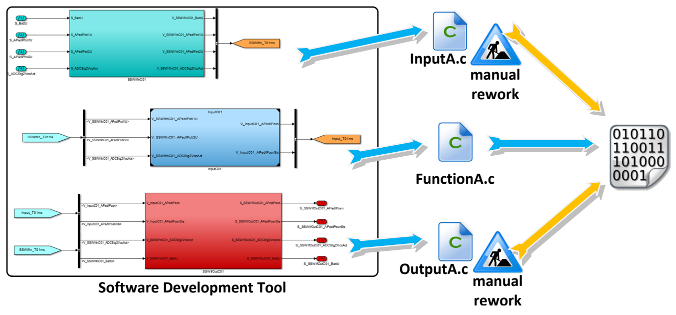


Fig. 1. ICC1 AUTOSAR Approach Methodology with Required Manual Intervention

II. Related Work

Development of automotive embedded software as well as the configuration of the underlying basic software and embedded systems are engineering domains and research topics aimed at moving the development process to an automated work-flow for improving the consistency and tackling the complexity of the software development process across expertise and domain boundaries. Recent publications are mainly based on AUTOSAR [1] methodology.

Due to the ever increasing software complexity of the last few years more and more efforts are becoming necessary to manage the development process of automotive embedded software. To handle this complexity the AUTOSAR consortium was founded and the AUTOSAR methodology provides standardized and clearly defined interfaces between different software components. The AUTOSAR approach features three different classes of implementation (ICC - implementation conformance class). The main benefit of the AUTOSAR ICC1 approach clearly relies on the time-saving in terms of no additional familiarization with usually very complex and time-consuming AUTOSAR tools compared to the full AUTOSAR approach (ICC3). The ICC1 approach does not take advantage of the AUTOSAR benefits from the full AUTOSAR tool-chain supporting tools for RTE configuration and communication interfaces, but standardized component interfaces for exchange of data between the ASW and BSW and therefore features the separation of application specific and hardware specific software parts (like native non-AUTOSAR development). ICC1 mainly focuses on SW engineering and more specifically on the integration of ASW into a given SW architecture. However, the aspects related to control systems engineering (including HW/SW co-design) are not integrated and aspects such as HW/SW interface definition must be performed manually, as indicated in Figure 1. The tool approach introduced in this work enhances this aspect by providing a framework for the visualization of both ASW and BSW interface configuration and automated generation of these interfacing $.c$ and $.h$ files (see Figure 2). Furthermore, the available hardware- software interfacing (HSI) information can be used to generate basic software (BSW) components configurations and the HSI information import functionality can also handle HSI spreadsheet templates to ensure more versatility of the tool.

An approach for an AUTOSAR migration of existing automotive software is described in the work of Kum et. al [10]. The authors highlight the benefits of separating the application software and the basic software and present an approach to configuration of basic software modules instead of time consuming and error-prone manual coding of embedded software. The automatic generation of automotive embedded software and the resultant configuration of the embedded systems thus improves quality as well as re-usability.

In [11], the authors describe a framework for a seamless configuration process for the development of automotive embedded software. The framework is also based on AUTOSAR which defines the architecture, methodology, and application interfaces. The configuration process is established via system configuration and ECU configuration. All the configurations and descriptions used are stored in separate XML (Extensible Markup Language) files, containing described and classified parameters and the associated information. The authors additionally specify a meta-model for the AUTOSAR exchange formats that describe the ECU configuration parameter definition and the ECU configuration description.

Jo et al. [9] describe an approach for the design of a vehicular code generator for distributed automotive systems. The increasing complexity during development of an automotive embedded software and systems and the manual generation of software have the effect of leading to more and more software defects and problems. The authors thus integrated a RTE module into their earlier development phase tool to design and evolve an automated embedded code generator with a predefined generation process. The presented approach saves time through automated generation of software code, compared to manual code generation, it reduces error-prone and time-consuming tasks and is also based on an AUTOSAR aligned approach. The output of the code generator tool is limited to the RTE source code and the application programming interface (API) of the input information. As in our approach, the configuration of software modules, is not focused.

Piao et al. [15] illustrate a design and implementation approach of a RTE generator for automotive embedded software. The RTE layer is located in the middle-ware layer of the AUTOSAR software architecture and combines the top layer mentioned as application software with the underlying hardware and basic software. Automated code generation aims at moving the development steps closer together and thus improving the consistency of the software development process. The output of the automated RTE generator are communication API functions for AUTOSAR SW components of the ASW.

Focusing on software complexity, Jo et al. [7] presents a design for a RTE template structure to manage and develop software modules in automotive industry. The authors focus on the design of a RTE structure also based on the AUTOSAR methodology. Within this design they describe the Virtual Functional BUS (VFB) which establishes independence between the Application Software (ASW) and the underlying basic software (BSW) and hardware.

In [14], an approach for realizing location-transparent interaction between software components is shown. The proposed work illustrates the relationship between the RTE and the VFB and shows which artifacts of the VFB are necessary for the generation of the RTE.

A work depicting the influence of the AUTOSAR methodology on software development tool-chains is presented by Voget [19]. The tool framework presented, named ARTOP (AUTOSAR Tool Platform), is an infrastructure platform that provides features for the development of tools used for the configuration of AUTOSAR systems. The implemented features are base functionalities required for different AUTOSAR tool implementations. The work does not, however, focus on a specific tool integration.

To summarize, none of the approaches described above supports (1) the generation of source code and (2) configuration of the basic software from information available at system level and from system models. The approach we present by contrast, supports not only the automatic generation of the RTE source code, but also the automated generation of basic software configuration of embedded systems from system models.

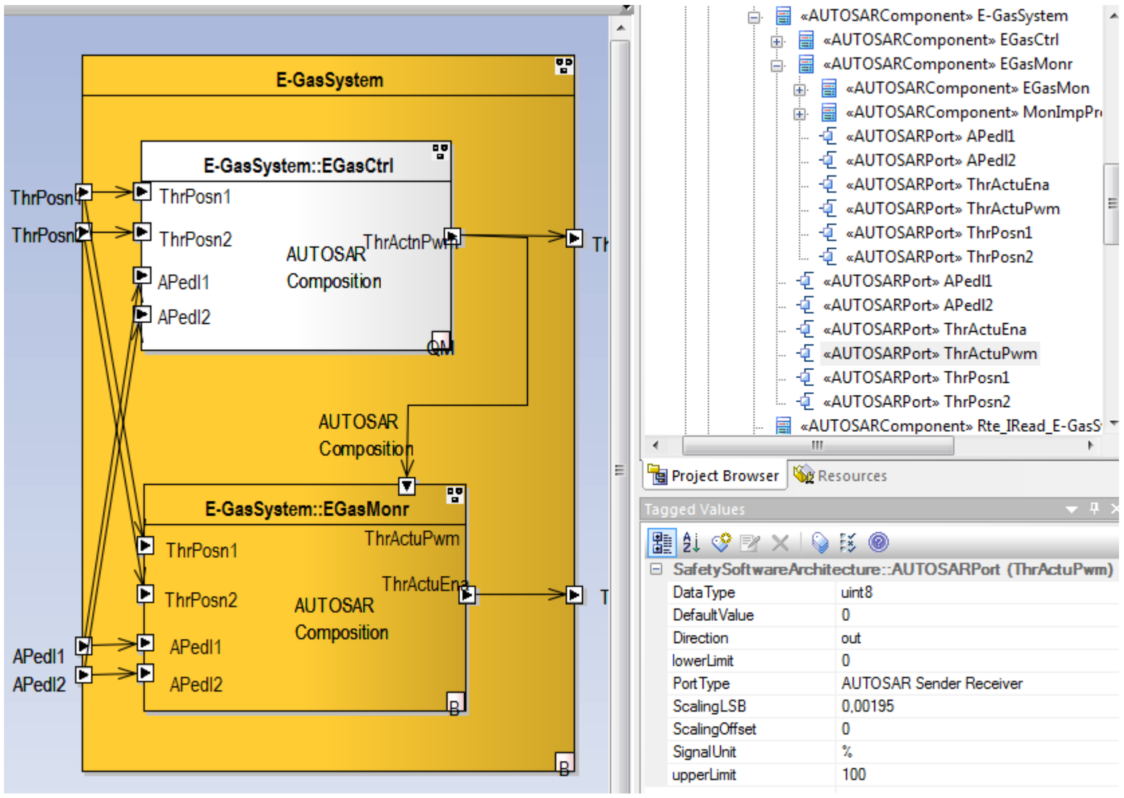
III. Basic Software Interface and Configuration Generation Approach

The underlying concept of the approach is to have a consistent information repository as a central source of information, to store all information of all engineering disciplines involved in embedded automotive system development in a structured manner [13]. The concept focuses on allowing different engineers to do their job in their own specific way, but providing traces and dependency analysis of features concerning the overall system, e.g. safety, security, or dependability. The approach stirs out of common AUTOSAR based approaches and additionally supports a non-AUTOSAR or AUTOSAR ICC1 approach, which are frequently hampered due to a lack of supporting tools. The decision of not fostering a full AUTOSAR approach is based on the one hand on focusing not only AUTOSAR based automotive software development and on the other hand, experiences we have made with our previous approach [12] confirm the problem mentioned by Rodriguez et al. [16]. Not all development tools fully support the entire AUTOSAR standard, because of its complexity, which leads to several mutual incompatibilities and interoperability problems.

Figure 2 shows an overview of the approach and highlights the main contributions. For a more detailed overview of the orchestration for the overall development tool-chain see [13].

The tool approach introduced in this work provides a framework for the visualization of ASW and BSW interface configuration and automated generation of these interfacing *.c* and *.h* files (see Figure 2). Furthermore, the available hardware- software interfacing (HSI) information can be used to generate basic software (BSW) component configurations and the HSI information import functionality can also handle HSI spreadsheet templates to ensure more versatility of the tool. More specifically, the contribution proposed in this work consists of the following parts:

- *AUTOSAR aligned UML modeling framework*:
  Enhancement of an UML profile for the definition of AUTOSAR specific artifacts, more precisely, for the definition of the components interfaces (based on the virtual function bus abstraction layer), see Figure 2 – HW and SW Modeling Framework.
- *BSW and HW module modeling framework*:
  Enhancement of an UML profile to describe BSW components and HW components. To ensure consistency of the specification and implementation for the entire control system, see Figure 2 – HW and SW Modeling Framework.
- *RTE generator*:
  Enables the generation of interface files (*.c* and *.h*) between application-specific and hardware-specific software functions, see Figure 2 – ASW/BSW Interface Generator .



«AUTOSARComponent» E-GasSystem
«AUTOSARComponent» EGasCtrl
«AUTOSARComponent» EGasMonr
«AUTOSARComponent» EGasMon
«AUTOSARComponent» MonImpPr
«AUTOSARPort» APedl1
«AUTOSARPort» APedl2
«AUTOSARPort» ThrActuEna
«AUTOSARPort» ThrActuPwm
«AUTOSARPort» ThrPosn1
«AUTOSARPort» ThrPosn2
«AUTOSARPort» APedl1
«AUTOSARPort» APedl2
«AUTOSARPort» ThrActuEna
«AUTOSARPort» ThrActuPwm
«AUTOSARPort» ThrPosn1
«AUTOSARPort» ThrPosn2
«AUTOSARComponent» Rte_IRead_E-GasS

SafetySoftwareArchitecture::AUTOSARPort (ThrActuPwm)

DataType	uint8
DefaultValue	0
Direction	out
lowerLimit	0
PortType	AUTOSAR Sender Receiver
ScalingLSB	0,00195
ScalingOffset	0
SignalUnit	%
upperLimit	100

Fig. 3. Screenshot of the SW Architecture Representation within the System Development Tool and Representation of the Interface Information

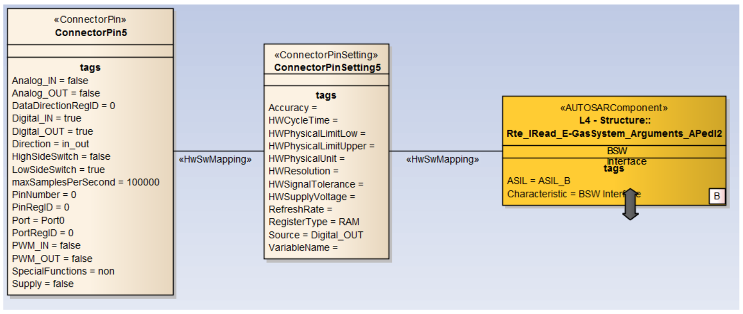
- *Basic software configuration generator*:
  Generates BSW configurations according to the specifications within the HSI definition, see Figure 2 – BSW Configurator.
- *Spreadsheet information importer*:
  Enables the import of HSI definition information done in spreadsheet format, see Figure 2 – Spreadsheet Information Importer.

This proposed approach closes the gap between system-level development of abstract UML-like representations and software-level development, also mentioned by Giese et al. [5], Holtmann et al. [6], and Sandmann and Seibt [17] by supporting consistent information transfer between system engineering tools and software engineering tools. Furthermore the approach minimizes redundant manual information exchange between tools and contributes to simplifying seamless safety argumentation according to ISO 26262 for the developed system. The benefits of this development approach are highly noticeable in terms of re-engineering cycles, tool changes, and reworking of development artifacts with alternating dependencies, as mentioned by Broy et al. [3].

The contribution proposed in this work is part of the framework presented in [13] aiming towards software development in the automotive context. The implementation of the approach is based on versatile C# class libraries (dll) and API command implementations to ensure tool and tool version in-dependence of the general-purpose UML modeling tool (such as Enterprise Architect or Artisan Studio) and other involved tools (such as spreadsheet tool and software development framework). The following sections describe those parts of the approach that make key contributions in more details.

A. *AUTOSAR aligned UML modeling framework*

The first part of the approach is a specific UML modeling framework enabling software architecture design in AUTOSAR like representation within a state-of-the-art system development tool (in this case Enterprise Architect). A specific UML profile to limit the UML possibilities to the needs of software architecture development of safety-critical systems and enable software architecture design in AUTOSAR like representation within the system development tool (Enterprise Architect). In addition to the AUTOSAR VFB abstraction layer [2], the profile enables an explicit definition of components, component interfaces, and connections between interfaces. This provides the possibility to define software architecture and ensures proper definition of the communication between the architecture artifacts, including interface specifications (e.g. upper limits, initial values, formulas). Hence the SW architecture representation within EA can be linked to system development artifacts and traces to requirements can be easily established. This brings further benefits in terms of constraints checking, traceability of development decisions (e.g. for safety case generation), reuse and ensures the versatility to also enable AUTOSAR aligned development as proposed in [12]. Figure 3 shows an example of software architecture artifacts and interface information represented in Enterprise Architect. As can be seen in the depiction, all artifacts required to model the SW architecture are represented and inherit the required information as tagged values.


```
«ConnectorPin»
ConnectorPin5
tags
Analog_IN = false
Analog_OUT = false
DataDirectionRegID = 0
Digital_IN = true
Digital_OUT = true
Direction = in_out
HighSideSwitch = false
LowSideSwitch = true
maxSamplesPerSecond = 100000
PinNumber = 0
PinRegID = 0
Port = Port0
PortRegID = 0
PWM_IN = false
PWM_OUT = false
SpecialFunctions = non
Supply = false
```

«HwSwMapping»

```
«ConnectorPinSetting»
ConnectorPinSetting5
tags
Accuracy =
HWCycleTime =
HWPhysicalLimitLow =
HWPhysicalLimitUpper =
HWPhysicalUnit =
HWResolution =
HWSignalTolerance =
HWSupplyVoltage =
RefreshRate =
RegisterType = RAM
Source = Digital_OUT
VariableName =
```

«HwSwMapping»

```
«AUTOSARComponent»
L4 - Structure::
Rte_IRead_E-GasSystem_Arguments_APedl2
BSW
Interface
tags
ASIL = ASIL_B
Characteristic = BSW Interface
```

Fig. 4. Screenshot of the BSW and HW Pin Representation within the System Development Tool

B. BSW and HW Module Modeling Framework

The AUTOSAR architectural approach ensures hardware-independent development of application software modules until a very late development phase and therefore enables application software developers and basic software developers to work in parallel. The hardware profile of the approach allows a graphical representation of hardware resources (such as ADC, CAN), calculation engines (core), and connected peripherals which interact with the software. Special basic software (BSW) and hardware module representations are assigned to establish links to the underlying basic software and hardware layers. This enables an intuitive graphical means of establishing software and hardware dependencies and a hardware-software interface (HSI), as required by ISO 26262. Software signals of BSW modules can be linked to HW port pins via dedicated mappings. On the one hand this enables the modeling and mapping of HW specifics and SW signals, see Figure 4 and on the other hand this mapping establishes traceable links to port pin configurations. A third point is that this HW dependencies can be used to interlink scheduling and task allocation analysis tools for analysis and optimization of resource utilization.

C. Runtime Environment Generator

The third part of presented approach is the SW/SW interface generator. This dll- based tool generates $.c$ and $.h$ files defining SW/SW interfaces between application software signals and basic software signals based on modeled HSI artifacts. In addition, this generation eliminates the need for manual SW/SW interface generation without adequate syntax and semantic support and ensures the reproducibility and traceability of these configurations.

Figure 5 shows the conceptual overview of generated files. The $.c$ and $.h$ files on application software level are generated via a model-based software engineering tool, such as Matlab/Simulink. The files on the basic software level are usually provided by the hardware vendor. While the files referred to in the SW/SW interface layer are generated by our approach.

The generated files are designed in a two-step approach. The first step of the interfacing approach ($interface.c$ and $interface.h$) establishes the interface between ASW and BSW based on AUTOSAR RTE calls. The second step ($AVLIL\_BSWa.c$ and $AVLIL\_BSWa.h$) maps these AUTOSAR RTE based calls to the HW specific implementation

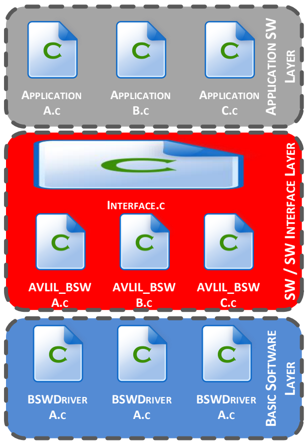


Fig. 5. Overview of Architecture Level Files Generated by the Interface Generator

of basic SW drivers. This ensures independence from implementation of the BSW drivers and also features an AUTOSAR ICC1 approach if needed.

D. Basic Software Configuration Generator

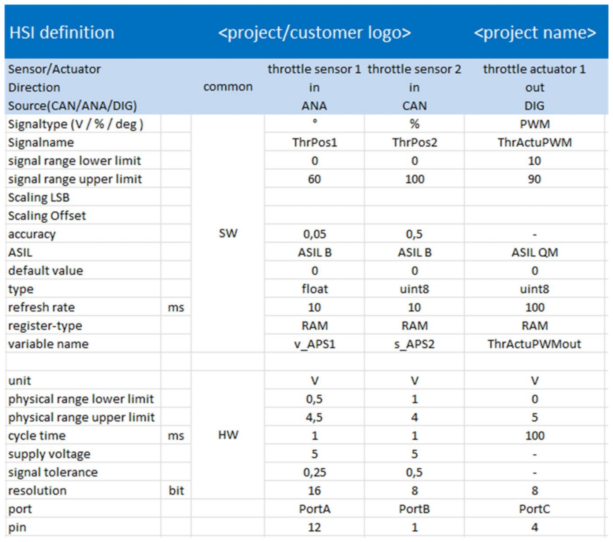
The basic software configuration generator is also part of the dll- based tool, which generates BSW driver specific $*\_cfg.c$ files. These files configure the vendor specific low-level driver (basic software driver) of the HW device according to the HSI specifications. The mapping of HSI specifications to low-level driver configuration is hardware and low-level driver implementation specific and needs to be done once per HW device and supported low-level driver package.

E. HSI Spreadsheet Information Importer

The HSI definition requires mutual domain knowledge of hardware and software and is to be a work product of a collective workshop of hardware, software, and system experts and will act as the linkage between different levels of development. Consistency and evidence of correct implementation of HSI can be challenging in case of concurrent HW and SW development and cross-dependencies of asynchronous update intervals. Therefore, this approach enables a practicable and intuitive way of engineering HSI definitions in a spreadsheet tool (Excel) and transforms them to a reusable and versionable representation in the MDB tool (Enterprise Architect). The spreadsheet template defines the structure of the data representation in a project-specific customizable way. On the one hand this enables a practicable and intuitive means of engineering HSI definitions with spreadsheet tools, while their machine- and human-readable notation ensures a cost- and time-saving alternative to the usually complex special-purpose tools, while on the other hand it enables the generation of SW/SW interface files and BSW configurations without the need for a model-based development toolchain in place. Figure 6 depicts the project-independent template structure for HSI definition data preparation.

IV. Application of the Proposed Approach

This section demonstrates the benefits of the introduced approach for development of automotive embedded systems.



HSI definition			<project/customer logo>		<project name>
Sensor/Actuator			throttle sensor 1	throttle sensor 2	throttle actuator 1
Direction		common	in	in	out
Source(CAN/ANA/DIG)			ANA	CAN	DIG
Signaltype (V / % / deg)			°	%	PWM
Signalname			ThrPos1	ThrPos2	ThrActuPWM
signal range lower limit			0	0	10
signal range upper limit			60	100	90
Scaling LSB					
Scaling Offset					
accuracy		SW	0,05	0,5	-
ASIL			ASIL B	ASIL B	ASIL QM
default value			0	0	0
type			float	uint8	uint8
refresh rate	ms		10	10	100
register-type			RAM	RAM	RAM
variable name			v_APS1	s_APS2	ThrActuPWMout
unit			V	V	V
physical range lower limit			0,5	1	0
physical range upper limit			4,5	4	5
cycle time	ms	HW	1	1	100
supply voltage			5	5	-
signal tolerance			0,25	0,5	-
resolution	bit		16	8	8
port			PortA	PortB	PortC
pin			12	1	4

Fig. 6. Example of a project-independent spreadsheet template structure for HSI definition

TABLE I. Overview of the Evaluation Use-Case SW Architecture

Object type	**Element-count**	**Configurable Attributes per Element**
ASW Modules	10	3
BSW Modules	7	3
ASW Module Inputs	54	10
ASW Module Outputs	32	10
ASW/ASW Interfaces	48	-
ASW/BSW Interfaces	19	-
HW/SW Interfaces	19	13

We used an automotive battery management system (BMS) as the use-case for the evaluation of the approach. This use-case is an illustrative material, reduced for internal training purposes and is not intended to be either exhaustive in scope or to represent leading-edge technology.

The definition of the software architecture is usually done by a software system architect within the software development tool (Matlab/Simulink). With our approach this work package is included in the system development tool (as shown in Figure 3). This does not hamper the work of the software architect but enables the possibility to also link existing HSI mapping information to the SW architecture (as shown in Figure 4).

The use-case consists of 10 ASW modules and 7 BSW modules with 19 interface definitions between ASW and BSW and makes use of the 3 fundamental low-level HW functions (digital input/output, analog input/outputs, and PWM outputs). A more complete overview of use-case is given in Table I.

The definition of the 19 HW/SW interfaces with 10 parameters for each SW signal and 13 parameters for each HW pin sums up to 437 parameter configurations within the HSI spreadsheet template or in the MDB tool, which can be used to generate ASW/BSW interfaces and BSW configurations.

This results in the file architecture depicted in Figure 7. With the use of the approach 8 additional interfacing files with 481 lines of code (LoC) source and 288 LoC configuration have been generated.

In terms of getting started with AUTOSAR aligned development or supporting non-AUTOSAR SW development our approach features a smooth first step approach of the ICC1 AUTOSAR and generates an interface layer (similar to AUTOSAR RTE) without relying on full AUTOSAR tooling support. In terms of safety-critical development the approach presented supports traceability links between BSW configurations to HSI information and eliminates the need of manual interface source code rework, which further surmounts the main drawbacks of the ICC1 AUTOSAR approach.

V. Conclusion

An important challenge for the development of embedded automotive systems is to ensure consistency of the design decisions, SW implementations, and driver configurations, especially in the context of safety-related development. This work presents an approach which seamlessly describes safety-critical software, from requirements at the system level down to software component implementation in a traceable manner. The available hardware- software interfacing (HSI) information can thus be used to generate basic software (BSW) component configurations, as well as automatic software interface layer generation (interface between application software and basic software). With this aim in mind a framework consisting of a basic software configuration generator and a software interface generator producing $.c$ and $.h$ files for linking ASW and BSW has been presented, which can also be used in combination with a spreadsheet based HSI definition. The main benefits of this enhancement are: improved consistency and traceability from the initial design at the system level down to the single CPU driver configuration, together with a reduction of cumbersome and error-prone manual work along the system development path. Further improvements of the approach include the progress in terms of reproducibility and traceability of configurations for software development (such as driver configurations and SW-SW interfaces).

The application of the presented approach has been demonstrated utilizing an automotive BMS use-case, which is intended to be used for training purposes for students and engineers and does not represent either an exhaustive or a commercial sensitive project. While the authors do not claim completeness of the analysis (due to confidentiality issues), the benefits of the approach are already evident.

Acknowledgments

This work is partially supported by the EMC^2 and the $MEMCONS$ projects.

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement nr 621429 (project EMC^2) and financial support of the ”COMET K2 - Competence Centers for Excellent Technologies Programme” of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ),

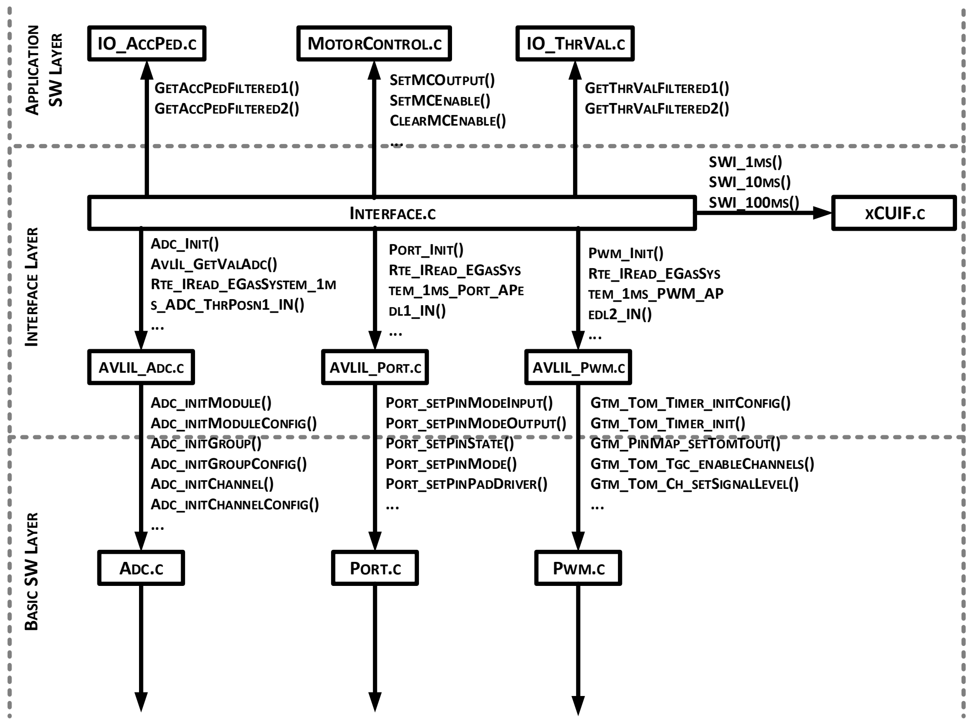


Fig. 7. Excerpt of Generated Files for the BMS Use-Case

the Austrian Research Promotion Agency (FFG), the Province of Styria, and the Styrian Business Promotion Agency (SFG).

Furthermore, we would like to express our thanks to our supporting project partners, AVL List GmbH, Virtual Vehicle Research Center, and Graz University of Technology.

References

[1] AUTOSAR development cooperation. AUTOSAR AUTomotive Open System ARchitecture, 2009.

[2] AUTOSAR Development Cooperation. Virtual Functional Bus. online, 2013.

[3] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-based Development: from Isolated Tool to Integrated Model Engineering Environments. *IEEE Magazin*, 2008.

[4] C. Ebert and C. Jones. Embedded Software: Facts, Figures, and Future. *IEEE Computer Society*, 0018-9162/09:42–52, 2009.

[5] H. Giese, S. Hildebrandt, and S. Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages pp. 555 –579, 2010.

[6] J. Holtmann, J. Meyer, and M. Meyer. A Seamless Model-Based Development Process for Automotive Systems, 2011.

[7] J. Hyun Chul, P. Shiquan, C. Sung Rae, and J. Woo Young. RTE Template Structure for AUTOSAR based Embedded Software Platform. In *Basic Research Program of the Ministry of Education, Science and Technology*, pages 233–237, 2008.

[8] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 1-10, 2011.

[9] H. C. Jo, S. Piao, and W. Y. Jung. Design of a Vehicular code generator for Distributed Automotive Systems. In *Seventh International Conference on Information Technology*. DGIST, 2010.

[10] D. Kum, G.-M. Park, S. Lee, and W. Jung. AUTOSAR Migration from Existing Automotive Software. In *International Conference on Control, Automation and Systems*, COEX, Seoul, Korea, 2008. DGIST.

[11] J.-C. Lee and T.-M. Han. ECU Configuration Framework based on AUTOSAR ECU Configuration Metamodel. 2009.

[12] G. Macher, E. Armengaud, and C. Kreiner. Automated Generation of AUTOSAR Description File for Safety-Critical Software Architectures. In *12. Workshop Automotive Software Engineering (ASE)*, Lecture Notes in Informatics, pages 2145–2156, 2014.

[13] G. Macher, E. Armengaud, and C. Kreiner. Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In *7th European Congress Embedded Real Time Software and Systems Proceedings*, pages 256 –263, 2014.

[14] N. Naumann. Autosar runtime environment and virtual function bus. Department for System Analysis and Modeling.

[15] S. Piao, H. Jo, S. Jin, and W. Jung. Design and Implementation of RTE Generator for Automotive Embedded Software. In *Seventh ACIS International Conference on Software Engineering Research, Management and Applications*. DGIST, 2009.

[16] E. Rodriguez-Priego, F. Garcia-Izquierdo, and A. Rubio. Modeling Issues: A Survival Guide for a Non-expert Modeler. *Models2010*, 2:361–375, 2010.

[17] G. Sandmann and M. Seibt. AUTOSAR-Compliant Development Workflows: From Architecture to Implementation - Tool Interoperability for Round-Trip Engineering and Verification & Validation. *SAE World Congress & Exhibition 2012*, (SAE 2012-01-0962), 2012.

[18] G. Scuro. Automotive industry: Innovation driven by electronics. http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/, 2012.

[19] S. Voget. AUTOSAR and the Automotive Tool Chain. In *DATE10*, 2010.

From system functional definition to software code

David Lesens
Airbus Defence and Space

Keywords: Model Driven Engineering, SysML, Automatic Code Generation, COTS, DSL

Abstract: This paper addresses the classical problem of system to software engineering following a Model Driven Engineering (MDE) approach. Even if this approach is now widely used in the industry, some issues remain: Long term availability of the tools (for projects with duration of several decades), use of standards and Commercial Of The Shelf (COTS) tools versus Domain Specific Language (DSL), different modelling tools for the system and the software, quality and mastering of automatically generated code. This paper shows how it is possible to take simultaneous benefit of COTS (low price), DSL (adapted to specific needs) and in-house tools (which can be maintained for very long periods of time) to develop complex critical systems.

1. Introduction

Since several years, Model based System Engineering (MBSE) has been shown efficient to improve the capture of system requirements. Airbus Defence and Space – Space Systems has for instance successfully deployed SysML modelling to support the functional definition of the avionics sub-system of a launcher such as Ariane 5 Mid-life Evolution ([4]) or of a spacecraft such as the European Service Module (ESM) of the Multi-Purpose Crew Vehicle (MPCV, [11]). The functional part of the system definition is formalised by a SysML ([18]) model developed in co-engineering by the system experts and the modelling experts. Several documents are thus partially automatically generated from this single model: The system Definition Files, the digital Interface Control Documents (ICD) and the software Technical Specification. MBSE improves the communication between the teams and improves the generated documentation quality.

This paper shows how Airbus Defence and Space – Space Systems has defined a process and a set of tools to extend this approach to the software development (Model Driven Engineering or MDE). The objective of this work is to generate from a single model shared between the system and the software the three previously mentioned documents and a part of the software code.

After this introduction, the section 2 will quickly present the system functional modelling today deployed for the development of space launchers. The section 3 will summarize previous studies on software modelling and explain why these approaches have finally not been selected for the development of future space launchers. The section 4 will describe the specific multithreading design used for space launchers and how it has been decided to model it. Finally, the section 5 will show how an important part of the flight software can be automatically generated from these models. The section 6 will compare the proposed solution with the ones presented in section 3 and the section 7 will present the conclusion.

2. System functional modelling

The avionics subsystem of a space launcher is mainly responsible for (1) the flight control (navigation, guidance and control) and (2) the mission management (ground / board protocol, ignition and stop of engines, release of stages…). It is made up of a centralised on-board computer hosting the flight software (or potentially several computers working in a redundant mode to ensure the fault tolerance) and a set of avionics sensors (gyroscopes, accelerometers…) and actuators (valves, pyrotechnic commands, electro-mechanical actuators…) linked by a communication network.

The system is then cyclically executed: Measurement acquisition, flight control algorithm, execution of commands by actuators, and so on. The needs of the flight control toward the launcher system is translated into a set of blocks (hardware or algorithmic) communicating through data-flows. This functional architecture is modelled by SysML Internal Block Diagrams (IBD). The content of the algorithm blocks is textually specified or modelled in Matlab and then coded in Ada.

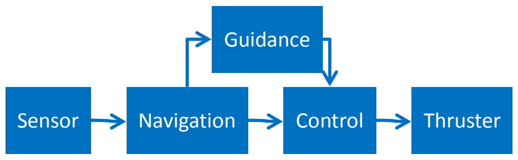


Figure 1: Functional data-flow architecture modelled in SysML IBD

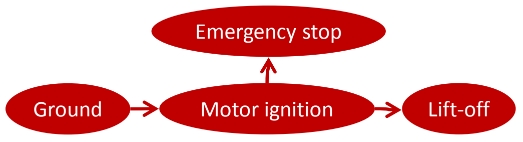


Figure 2: Part of a launcher mission described by a finite state machine in SysML

The mission management describes the acyclic behaviour of the system. The phases of the launcher mission (ground phase, motor ignition and then lift-off or emergency stop in case of failure) are thus modelled by finite state machines in SysML and by a textual DSL (Domain Specific Language) adapted to the description of a space launcher mission.

The use of a COTS modelling tool (e.g. Rhapsody [14]) allows decreasing the costs: COTS tools are generally very mature and require only light customization. However, one of the main drawbacks of COTS is the difficulty of maintenance for a very long period of time. The use of a widely used standard ensures that editors will always be available in the future.

SysML has thus been selected because it is a standard widely used in the industry and in the academic world and because it provides the two kinds of diagrams needed to describe the functional architecture and the flight software of space launchers: Internal Block Diagrams and Finite State Machines.

The semantics of the modelling (SysML + DSL) is the one of Synchronous Languages such as Lustre ([8]) or Signal ([17]):

- The system is cyclically activated at a constant frequency.
- The inputs of the system are supposed to be available at the beginning of each cycle of execution.
- The outputs computed by the system are provided at the end of the cycle.

This approach is well known to ensure a full determinism of the system and allow simple compositions of elements with a predictable memory usage.

Here is an example of the DSL describing the execution in parallel of two commands and inspired by the notion of On Board Control Procedure (OBCP, [12], also called “functional sequences”):

```
sequence Lift-off is
        fork Cmd_Pyro;
        wait 5 ms;
        Valve_Cmd;
        wait end of Cmd_Pyro;
        Start_Control;
end;
```

Figure 3: DSL describing a functional sequence

This example is valid with respect to the synchronous semantics only if the duration “5 ms” is a multiple of the basic cycle of the system.

The communication network (related to the avionics system design) and the middleware and the threads (related to the software real-time design) are abstracted in this phase of development.

3. Previous studies on software design modelling

In parallel to the deployment of SysML modelling to capture the system functional definition, several modelling approaches have been assessed in order to capture the software design and to potentially generate automatically part of the flight software:

- SCADE Suite ([16]) was a promising track thanks to its certified code generator ensuring a high quality of the generated code. It has however several limitations
    - The SCADE modelling is partially redundant with the SysML modelling (especially for what concerns the finite state machine and the functional architecture). The development cost of these two models removes the benefits of the approach

  - SCADE does not model today the multithreading architecture. Developing multithreading software with SCADE requires thus the independent modelling of each thread in SCADE and then the manual development of a real-time sequencer responsible for the activation of each thread.

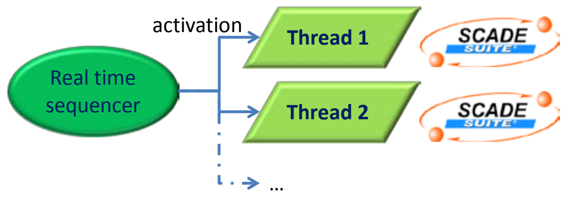


Figure 4: Multithreading modelling in SCADE

    The SCADE model becomes then a design model distinct from the functional model. This decreases greatly the advantages of the modelling
  - The code generated from SCADE has not been designed to be manually maintained. This implies the need to maintain the SCADE modelling tool and the code generator during the complete life of the project (up to 30 years for a space launcher). The cost of this maintenance removes on the long term the saving of automatic code generation
- AADL ([1]) allows modelling the avionics and the software multithreading architecture. As for SCADE, this modelling is partially redundant with the SysML modelling selected for the system functional description. Moreover, AADL is able to capture any kind of software multithreading architecture, even the more complex. The flight software of a space launcher being critical, its multithreading architecture is very simple and based on the Rate Monotonic Scheduling (RMS [15], see section 4). AADL is thus too complex for the needs.
- MARTE ([9]) has the advantage of being close from SysML and UML. However, it has the same drawbacks than AADL.
- UML ([19]) is a modelling language especially well-suited for object oriented software design. As the flight software of future launchers will be partially developed in an object oriented manner (with the Ada 2012 [3] programming language), UML seems a good choice. However, the object oriented approaches of UML and Ada 2012 are not fully compatible (see [2]). For instance:
  - The UML notion of “class” is replaced in Ada by the notions of “package” (to define the naming space) and of “tagged type” (to define the inheritance)
  - Ada has a restricted set of “visibility” compared to UML
  - …

  Moreover, as for SCADE, the UML modelling is partially redundant with the SysML modelling. It has thus been decided that UML models will be developed to describe the principle of design, but not for the detailed design (except for some specific parts) and will not be used for automatic code generation.

As a conclusion of these studies, it appears that all the studied software modelling languages have two drawbacks: The redundancy with SysML (selected for the system functional description) and the cost to maintain the toolsets during a long period of time. Airbus Defence and Space has thus decided to rely mainly on SysML completed by a DSL (Domain Specific Language) to model the design of launcher’s flight software (the same DSL being used for the system functional behaviour (see section 2) and the software design).

4. Software design modelling

Defining a DSL requires an accurate definition of the objectives of the modelling. This section describes the flight software design used today on space launchers and which has been the basis to define the DSL.

The software design is composed of the static design (definition of the hierarchical architecture of the software, definition of classes and of objects, definition of the interfaces of components) and of the real-time design (definition of threads, of their scheduling, of the communication between the threads and of the communication between the software and the communication network). The main drivers of the flight software design are:

- The functional and real-time determinism ensuring the representativeness and the reproducibility of the qualification tests
- The decrease of the development and validation costs

The determinism of the multithreading design is ensured by using an extension of a Rate Monotonic Scheduling:

- The software is composed of cyclic threads (no acyclic threads)
- Each thread has a period multiple of the period of the thread just faster (harmonic threads)

- The scheduling is pre-emptive with constant priority (a thread with a shorter period has the priority over a thread with a longer period)
- The communications between the threads are performed during time triggered rendezvous

The Figure 5 provides the example of two communicating threads (a slow one in blue and a quick one in yellow).

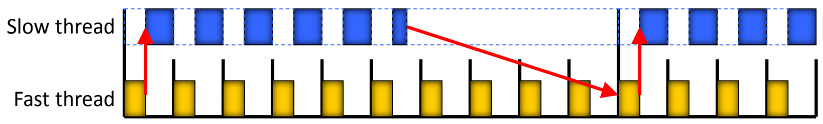


Figure 5: Time triggered communication between threads

The period of the slow thread is a multiple of the one of the quick thread. The quick thread has the highest priority. It can pre-empt the execution of the slow thread. The communications between the two threads are performed at the end of the period of the slow thread (in a protected section): even if the execution of the slow thread is quicker than expected, the communication will be performed at the same date. Thus, provided that each thread respects it worst case execution time, this design ensures the strict determinism of the software behaviour.

The decrease of the development costs is achieved by mapping the functional architecture and the software static design. This result is obtained by a co-engineering work between the system team (responsible for the functional architecture, see section 2) and the software team. The functional architecture shown Figure 1, which defines a set of functions and their communications, is thus directly used to generate the code (see section 5).

The work remaining at software design level is thus:

- To define the set of threads and their periods
- To map each function on a thread. Depending of the reactivity needs, a function at 10Hz can for instance be executed either at each cycle on a thread at 10Hz or at one cycle among two on a thread at 20Hz.

A simple DSL has been defined to describe this software design choices.

```
thread T1 is
        period (100 ms);
        functions (F1; F2);
end;
```

```
thread T2 is
        period (50 ms);
        functions (
                when 0 => (F3; F4);
                when 1 => (F3));
end;
```

Figure 6: Multithreading architecture described by a DSL

On the example of Figure 6:

- A major frame of 100 ms has been defined
- F1 and F2 are executed at 10 Hz (on the thread T1)
- F3 is executed at 20 Hz (on the thread T2)
- F4 is executed at 10 Hz (one cycle among two on the thread T2)

This real-time design is an extension of the synchronous language approach (on which the functional modelling described section 2 is based). The DSL describing the functional view (section 2) and the one describing the real-time design (section 5) rely thus on the same paradigm and are thus naturally compatible.

5. Automatic code generation

The multithreading design of the software (the definition of threads and the mapping of functions on the threads) remains today a manual activity which is formalised by a dedicated model using the DSL. All the remaining coding activities related to this multithreading design have been automated in an in-house tool generating Ada 2012 code from the SysML model and the DSL.

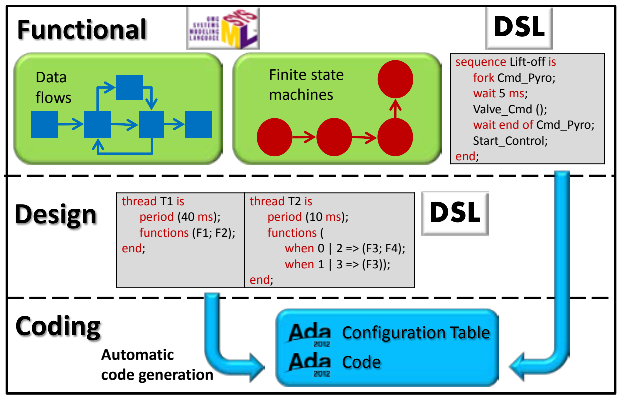
This automatic generation of code relies on a generic reusable library implementing:

- A scheduler of threads and of applicative software elementary blocks,

- An interpreter of a mission management (described by finite state machines and the DSL shown Figure 3).

Considering this generic reusable library, the following artefacts are automatically generated by the in-house tool:

- Configuration tables in Ada 2012 for the generic reusable library from the SysML finite state machine and the DSL describing the functional sequences (Figure 3) and for the definition of threads (Figure 6).
- Ada 2012 code for the skeleton of the functions (Figure 1) and the scheduling of these functions (Figure 6).



```
sequence Lift-off is
   fork Cmd_Pyro;
   wait 5 ms;
   Valve_Cmd ();
   wait end of Cmd_Pyro;
   Start_Control;
end;
```

```
thread T1 is
   period (40 ms);
   functions (F1; F2);
end;
```

```
thread T2 is
   period (10 ms);
   functions (
      when 0 | 2 => (F3; F4);
      when 1 | 3 => (F3));
end;
```

Figure 7: Code and configuration table automatic generation

There is no certification for space launchers similar to certification for civil aircraft. However, the quality of generated code remains a major issue: a failure of the flight software implies generally the loss of the launcher and of its payload (the satellites to be put in orbit) and in the worst possible case the destruction of the launch pad. The ECSS-E-ST-40C ([7]), the standard applicable to the development of space software in Europe, requires the classical reviews (of specification, of design, of code…) and test activities. The in-house code generator being not qualified according to this standard, reaching the same level of quality than for a manual code forbids suppressing any of these V&V activities. Reviews of code are for instance performed on the generated code.

The obligation of performing such reviews is often considered as a reason for not using automatic code generation because (1) the generated code is not enough readable to perform such review and (2) even a light modification in the source model may imply huge modifications in the generated code which makes mandatory a complete review of the generated code for each evolution. These two aspects were major requirements for the development of the in-house code generator: (1) the generated code is strictly equivalent of a manually written code and (2) the impact of a local model modification has a local modification on the generated code (allowing performing efficient comparisons between two versions of generated code and thus simplifying the code review).

The long term availability of the specific SysML modelling tool remains also an issue which has been solved in two ways: First, the notions of data-flows diagrams and of finite state machines have been also defined in the textual DSL; second, the code generator has been designed in order that the generated code is equivalent to a manual code, meaning that it can be easily manually maintained. During the development phase of a project, the SysML modelling tool will thus be used to decrease the development cost. After some years (5 to 10 years), i.e. during the maintenance phase, this tool will be potentially not any more maintained by the tool provider; the code will then be automatically generated from the textual DSL generated from the SysML model. If it is decided to not maintain any more the in-house code generator, it will still be possible to maintain manually the generated code.

The Figure 8 shows this three phases of maintenance:

1. The SysML model and the DSL are both maintained. The code is automatically generated. A DSL corresponding to the SysML model is generated for backup.
2. The SysML model is not maintained any more. The initial DSL and the DSL previously generated from SysML are maintained. The code is automatically generated.
3. The code generator is not maintained any more. The software code is manually maintained.

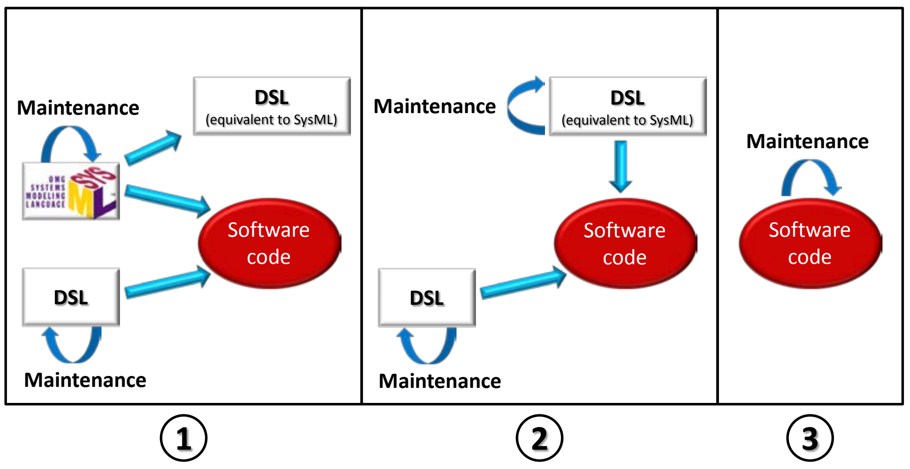


Figure 8: The three phases of maintenance

6. Synthesis

The following table summarizes the advantages and drawbacks of different modelling approach.

Approach	Functional view			Multithreading view	Safety	Long term maintenance
	Finite State Machine	Data flow	Functional sequences			
SCADE	Yes	Yes	At a low level of abstraction with finite state machine	No	Certified code generator	Rely on a proprietary tool
AADL	Yes but textual	More service oriented	No	Yes, but too complex for an extended RMS approach	No certified code generator	Standard
MARTE	See UML	See UML	See UML	Yes, but too complex for an extended RMS approach	No certified code generator	Standard
SysML (alone)	Yes but asynchronous	Yes (IBD)	Yes, but not formal enough	No	No certified code generator	Standard
UML	Yes, but asynchronous	No	Yes, but not formal enough	See MARTE	No certified code generator	Standard
SysML + DSL	Yes in SysML, with an adapted synchronous semantics	Yes in SysML (IBD) with an adapted synchronous and multithreading semantics	Yes in the DSL with the required synchronous semantics	Yes in the DSL, with an extended RMS approach	No certified code generator, but the automatically generated code is strictly equivalent to a manually written code	Standard SysML In-house tool The generated code can be manually maintained

7. Conclusion

Refining a system model to a software model and then to code remains an issue for critical system with a long duration: the tools are not maintained for the whole duration of the project and have generally a too large scope to generate code which can be manually maintained.

This paper has presented a solution taking benefit of the available modelling tools during the development, able to go smoothly from system functional definition to the code. This approach relies on the use of COTS (Commercial Of The Shelf) associated to DSL (Domain Specific Language) and in-house tools:

- The COTS tools are used to take benefit from mature and already widely deployed technologies
- DSL and in-house tools are used to take into account the specificities of the field

The quality and the long term maintenance issues are tackled by a specific effort on the code generator: the generated code is strictly equivalent to a manual code, can be reviewed and manually maintained.

The next step of improvement will be the harmonization of toolsets between the functional analysis (for instance with the use of the MEGA tool [10]), the system functional definition and the software (for instance with the Rhapsody tool [14]) and the physical architecture (for instance with Capella ([6]).

8. References

[1] AADL standard (www.aadl.info)

[2] "From Ada 83 to Ada 2012", Philippe Gast and David Lesens (Invited presentation at Ada Europe 2015)

[3] Ada Reference Manual (ISO/IEC 8652:2012(E))

[4] Ariane 5 program (http://www.arianespace.com/launch-services-ariane5/ariane-5-intro.asp)

[5] Ariane 6 program (http://www.airbusafran-launchers.com/home/#ariane)

[6] Capella tool (https://www.polarsys.org/capella)

[7] European Cooperation for Space Standardization (ECSS, http://www.ecss.nl/)

[8] Lustre (http://www-verimag.imag.fr/Lustre-V6.html)

[9] MARTE standard (www.omg.org/spec/MARTE)

[10] MEGA tool (www.mega.com)

[11] MPCV program (http://www.nasa.gov/exploration/systems/mpcv/index.html)

[12] Standard ECSS-E-ST-70-01C ("Spacecraft on-board control procedures")

[13] Ravenscar profile, see Ada Reference Manual [3]

[14] Rhapsody tool (http://www-03.ibm.com/software/products/en/ratirhap)

[15] Rate Monotonic Scheduling (Liu, C. L.; Layland, J. (1973), "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM 20)

[16] SCADE Suite tool (www.esterel-technologies.com/products/scade-suite)

[17] Signal (http://www.irisa.fr/espresso/home_html)

[18] SysML standard (www.omg.org/spec/SysML/1.3)

[19] UML standard (www.omg.org/spec/UML)

Session 14

Multicore & Predictability

Thursday 28th, 11:45 – Guillaumet

Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP)

Javier Jalle†,*, Mikel Fernandez†, Jaume Abella†, Jan Andersson⋆,
Mathieu Patte∓, Luca Fossati§, Marco Zulianello§, Francisco J. Cazorla†,‡

†Barcelona Supercomputing Center, Spain
*Universitat Politècnica de Catalunya, Spain
§European Space Agency, The Netherlands
⋆Cobham Gaisler, Sweden
∓Airbus Defence and Space, France
‡Spanish National Research Council (IIIA-CSIC), Spain

***Abstract*—The Space industry, as several other real-time industries, is assessing the use of multicore processors as their main computing platform. While multicore processors bring the potential of integrating several software (mixed-criticality) functions, their use also brings some challenges. In particular, tasks running in multicores may experience high contention delays when accessing multicores' shared resources. This makes that the load that a task puts on shared resources impacts the Execution Time Bounds[1] (ETBs) derived for other corunning tasks. In this paper we focus on the Cobham Gaisler NGMP – acknowledged as one of the multicore processors currently assessed by the European Space Agency for its future missions – for which we propose a measurement-based approach to bound contention interference. Given a task τ, instead of providing ETBs for the highest contention that any set of corunners can generate – already shown to be potentially high – our approach provides bounds that factor in the number of requests contenders generate regardless of how they align with τ's requests. This provides a good balance between ETBs accuracy and independence from the corunners, since our approach only requires controlling the number of requests each task makes to the shared resources.**

***Index Terms*—WCET, multicore, COTS, real-time**

I. Introduction

Real-Time Embedded systems are facing an increase in their performance demands across several domains, such as space, avionics and automotive, as a way to provide more value-added functionality. In the space domain, computing power requirements and the amount of data to be handled by on-board software is rising [30] due to the fact that space missions are becoming more autonomous. In this context, multicore processors can provide the performance required, while enabling the consolidation of applications[2] subject to different criticality levels, resulting in an overall reduction in power, space and weight. In the space domain the Next Generation Microprocessor (NGMP) architecture [5], whose latest implementation is the GR740 [6], is an architecture considered by the European Space Agency (ESA) for its future missions.

[1]We use Execution Time Bound (ETB) instead of Worst-Case Execution Time (WCET) estimate to refer to the upper-limits derived for tasks execution time in multicore. The reason is that WCET estimates, as they are commonly understood, establish a single value that upperbounds program's execution time under any circumstance. While this can be asserted for single-core simple architectures, this is not the case for multicores using more complex pipelines.

[2]In this paper we use the terms application and task interchangeably.

Multicores also bring their specific issues to the real-time domain among which contention in the access to hardware shared resources is one of the most prominent [2]. Uncontrolled contention makes that the execution time and Execution Time Bounds (ETB) derived for a task depend on the load its corunner tasks put on hardware shared resources, thus affecting time composability [21], which states that the ETB derived for a task in isolation should not be affected by the rest of the tasks running on the system. Time composability is a premise in many real-time designs since it enables (in the timing domain) incremental development and verification in integrated systems such as Integrated Modular Avionics, IMA [26] in avionics. During system development, time composability enables incrementally integrating applications without the need of regression tests to validate the timing properties of already-integrated applications, which heavily reduces integration costs. During operation, time composability enables updating functions and their associated software without the need for re-analyzing and re-qualifying the system. This is specially beneficial in domains like space where systems may operate during dozens of years and whose functionality is usually updated after deployment.

In a time-anomaly [24] free system, time composability can be achieved by modeling at analysis time a scenario in which each access that the task under analysis (τ) makes to a hardware shared resource suffers the highest contention possible. For instance, in the case of a round-robin bus accessed by Nc cores this is equivalent to assuming that each request suffers maximum contention from each of the remaining $Nc-1$ cores. That is, the single-access maximum (contention) delay, or $samd$, corresponds to:

$$samd = (N_c - 1) \times L_{bus} \qquad (1)$$

where L_{bus} is the maximum bus latency for a single request. The resulting ETB estimate in this scenario is *fully time composable* since it accounts for the maximum load that corunner tasks of τ can put (at operation time) on the target resource. This, though, comes at the cost of inflated ETB estimates (e.g. up to more than 5x times in a 4-core processor as reported in [12]). Tighter ETB estimates can be obtained by adjusting the bounds to the actual load that corunner tasks put on the target resource, which can be abstracted with an arrival curve [27]. However, time-composability is lost since the ETB for a task becomes dependent on its particular corunners. This confronts industry with the choice of time-composable inflated estimates or tighter non time-composable estimates.

In this paper we use measurement-based timing analysis, which a large fraction of safety-related systems resort on [31] – including the space industry. We propose a contention-prediction model that captures the effect of contention in the NGMP shared resources. For a given task, τ, our model enables deriving both fully time-composable bounds to the contention delay suffered by τ or partially time-composable bounds [11] *which depend on the number of requests generated by τ's corunner tasks, N_{req}, but not on how they align with τ's requests*. Derived bounds are valid for different corunner tasks as long as they generate at most N_{req} requests.

Our approach is motivated by the fact that, while the number of requests that a task generates can be bound with existing tools like Rapita System's Verification Suite (RVS) [23], how τ's and its corunners' requests interleave is hard, if at all possible, to measure and control. Hence, instead of predicting request interleaving, our approach derives contention delays for the worst-possible time-alignment of requests. The main contributions of this paper are as follows:

1) We make an in-depth analysis of the hardware shared resources in the NGMP, the way in which requests interact and the delay they may suffer on those resources.

2) We present a prediction model for the contention delay in the bus and the memory controller in the NGMP. Our model, which depends on the time requests take to access shared resources, deals with the case when there are several types of accesses to a resource and each type causes and suffers a different delay depending on the contending accesses. For instance, in the processor AMBA AHB bus, loads missing in the L2 take shorter than loads hitting in L2. We show how our model handles this case.

3) We evaluate our proposal in a solid setup comprising the GR740 implemented in a FPGA. Our proposal provides tighter ETBs than the fully time-composable proposal in [12], since it adapts to the contenders' load on shared resources in a still partially time-composable and friendly way.

The rest of this paper is organized as follows: Section II presents the related work. Section III provides information on the GR740. Section IV details our prediction model. Section V assesses the accuracy of our model. Finally, Section VI presents the main conclusions of this paper.

II. Related Work

Contention on the access to hardware shared resources has been thoroughly studied in the state of the art. A taxonomic summary of the relevant works can be found in [8]. Several techniques propose means to upper-bound, during the analysis phase of the system, the $samd$ that a task may suffer on the bus or in memory. In that line, hardware support has been proposed (though not yet implemented in any architecture we are aware of) to artificially delay each request a given τ does by $samd$ cycles [18][13][28]. Other approaches derive $samd$ by using a software-only approach: τ is run against a set of resource stressing kernels that put high load on the resource [12][9] making τ's requests suffer high contention delays.

Other techniques like those in [25] for buses rely on detailed information about resource access latencies and arbitration policies to derive $samd$. Other works, due to lack of information in the processor documentation derive $samd$ from measurements and feed it into static timing analysis. In particular [17] applies this approach to analyze the impact of contention in the P4080. $samd$ can also be derived for memory with [19][1] or without [15] hardware support.

In this paper we follow the theoretical approach in [10] that proposes a methodology to obtain the resource access 'profile' of a given task that defines the use of resources that the task makes on a target shared resource. That profile is used to derive the contention tasks suffer and generate when accessing that resource. In this work, which is a collaboration of end-users in the Space domain (Airbus Defence and Space and ESA), hardware technology providers (Cobham Gaisler) and a research institution (Barcelona Supercomputing Center) we assess the benefits of such an approach on a real platform, the GR740 addressing issues related to NGMP specific arbitration policies and access types to the different resources.

Finally, is it worth mentioning that with few exceptions [29][3], cache partitioning is the common solution in the context of CRTES due to the complexity of estimating ETB accurately on top of shared caches. In the case of the GR740, hardware support exists for way-partitioning the L2 cache. We enable this hardware feature in our experiments.

III. NGMP

The NGMP is being assessed as the processing platform by the ESA in its future missions. The NGMP is a quad-core processor system-on-chip based on the LEON4 SPARC V8 architecture [5] connected by a shared on-chip AHB processor bus to a shared L2 cache and memory, see Figure 1. The NGMP comprises 16 Performance Monitoring Counters (PMC) that can be configured with different events, providing support to measure access counts such in a way that it facilitates the implementation of our prediction model, more details are provided in Section IV-C. This section provides details on some aspects related to the contention in the access to NGMP's shared resources.

A. AHB Processor Bus

The AMBA AHB bus connects cores to the L2 cache and the I/O bridges[3]. The first consideration to make in the case of the bus is that there are different types of requests that can generate different inter-task contention: bus reads (loads) that

[3]In this work, we do not consider I/O related activities, which we assume managed at software level, so that only accesses to L2 interfere each other.

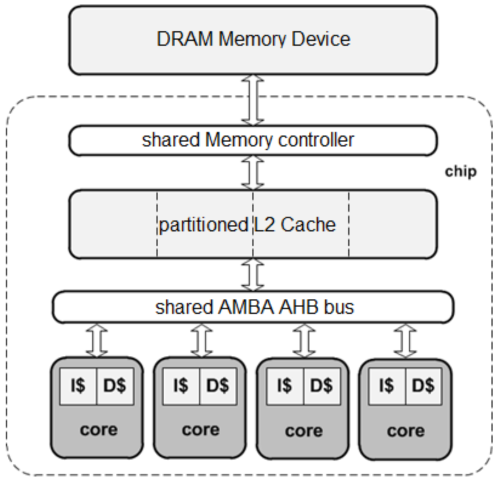


Fig. 1: Block diagram of the main elements of the NGMP

either hit (*l2h*) or miss (*l2m*) on the L2 cache and bus writes (stores) that either hit (*s2h*) or miss (*s2m*) on the L2 cache. These accesses behave differently because hits hold the bus while they are served. Instead, misses wait on a miss queue and are split, i.e. the L2 cache releases the bus while processing the miss, so that other cores can use the bus. In the NGMP, the AMBA AHB bus implements round-robin arbitration.

B. L2 Cache

In our experiments we use the master-index feature of the NGMP that partitions the L2 assigning one L2 cache way to each core. Hence, a given core suffers no contention interference in the L2 due to other cores' evictions.

Each of the request types identified before (*l2h*, *l2m*, *s2h* and *s2m*) has its own L2 access latency. Interestingly, the latency of requests of the same type can be variable. That is, for each request type access there is a Best-Case (BC) and a Worst-Case (WC) latency. This jitter is caused by the type of previous requests, despite they belong to a different task and hence go to a different cache partition. Our model takes this effect into account by assuming that all latencies suffered on the experiments have the BC and when computing the contention bounds, we add a correcting value that adds for each L2 access the corresponding difference between the WC and the BC. This adds pessimism but its advantage is two-fold: it is a safe upperbound and it removes the need to track the sequence of accesses to determine their exact latency.

The WC and BC latencies are obtained from table 40 in [7] and are 8, 13, 6 and 7 for *l2h*, *l2m*, *s2h* and *s2m* respectively in WC and 5, 6, 0 and 0 for BC.

C. Memory Controller

The memory controller acts as an interface between the processor and the DRAM memory. We differentiate two types of request in the memory: read and write. According to the DRAM protocol, each request has a latency to be responded depending on whether it is a read or write request respectively. The latency it takes the memory to go back into idle state, once a request starts being processed, is fixed regardless of whether the request is read or write and corresponds to the time till a new request can be processed. For this paper, we assume that the memory controller behaves as a FIFO queue. This is a simplification that helps upper bounding the memory controller latency though it introduces some pessimism. Providing a more accurate model of the memory is part of our future work.

IV. Prediction Models

Our prediction models use measurement-based timing analysis techniques to derive a multicore ETB (ETB_{mc}) for a task τ_i, given its ETB in isolation (ETB_{isol}). To that end, the models predict the total effect of contention in the access to the multicore hardware shared resources, called Contention Delay Bound (CDB), and add it to the ETB in isolation:

$$ETB_{mc} = ETB_{isol} + CDB \tag{2}$$

In order to derive CDB, we add the contribution of each hardware shared resource r, CDB_r:

$$CDB = \sum_{r \in R} CDB_r \tag{3}$$

To derive CDB_r, we upper-bound the maximum latency that every access from τ_i to r, n_i^r, may suffer from requests generated by τ_i's corunner tasks, referred to as $c(\tau_i)$.

CDB_r for τ_i assumes that each $\tau_j \in c(\tau_i)$ performs at most a given number of accesses (n_j^r) to resource r. Therefore, ETB_{mc} estimate for τ_i is composable with any other task $\tau_j' \in c'(\tau_i)$ as long as it performs fewer accesses ($n_j'^r$) to the shared resource than $\tau_j \in c(\tau_i)$:

$$n_j'^r \leq n_j^r \tag{4}$$

A. Bus Prediction Model

The NGMP comprises three main shared resources in its data path: the bus, the L2 cache and memory. Since the L2 can be partitioned we do not consider contention of the different tasks in the L2. We start by predicting CDB_{bus} for the bus and later apply the same approach for memory.

We explain three different ways of upper-bounding CDB_{bus}, which present represent different trade-offs between information required, such as the number of accesses of each corunner task, and tightness of the produced bound.

A.1. Theoretical Upper-Bound Delay (UBD)

In this reference model, based on [18], we assume that every single τ_i request is delayed by a request from each of the $N_c - 1$ contenders and that contending requests cause the highest delay, L_{bus}. This is the maximum contention scenario in round-robin arbitration, where the upper-bound delay a request can suffer is given by:

$$samd = (N_c - 1) \times L_{bus} \tag{5}$$

Hence, for τ_i with b_i accesses to the bus, CDB_{bus} is presented in Equation 6, where L_{bus} is the maximum delay any interfered request can suffer from a single interfering request.

$$CDB_{bus} = b_i \times samd = b_i \times (N_c - 1) \times L_{bus} \qquad (6)$$

Since we have four different types of requests with different latencies: l_{l2h}, l_{l2m}, l_{s2h} and l_{s2m}:

$$L_{bus} = \max\left(l_{l2h}, l_{l2m}, l_{s2h}, l_{s2m}\right) \qquad (7)$$

This model is time-composable by definition because it assumes that all b_i are interfered by i) the highest impact request from ii) all corunners. These two assumptions are sources of pessimism that enable full time-composability.

Interestingly in this model, the worst alignment among the requests of τ_i and the requests of its corunners is assumed. In reality, it can be the case that some τ_i requests become ready to be sent to the bus when its contenders requests have been partially processed so that each τ_i request suffers a delay smaller than L_{bus}. However, predicting how this alignment of requests can happen at operation time is hard (if at all possible). Any small shift in the execution of tasks can change it. Hence, this and the following models, provision time in CDB_{bus} for the worst-case alignment of requests.

A.2. Single-type Model

Analogously to the previous model, the one presented in this section assumes that every corunners' request causes a delay of L_{bus} on τ_i. Unlike the previous model, this one takes into account that not all τ_i requests might be interfered by one request of its corunner tasks. This usually happens when corunner tasks have fewer accesses than τ_i.

Let b_j be the number of accesses to the bus that each contender task $\tau_j \in c(\tau_i)$ performs. Given that tasks have different number of accesses, not all of them can interfere each other. In particular, for a given interfering task τ_j running in core j, *in the worst-case* only the minimum between the number of accesses of τ_i, b_i, and the number of accesses of the interfering task, b_j, suffer a contention delay of L_{bus}. That is, no more than b_i accesses can be interfered and no more than b_j can interfere. In order to compute the contention on the bus for task τ_i, we add the contribution of each interfering task τ_j:

$$CDB_{bus} = \sum_{\tau_j \in c(\tau_i)} \min\left(b_i, b_j\right) \times L_{bus} \qquad (8)$$

A.3. Multiple-type Model

The previous model assumes that each interfering request, i.e. those generated by $c(\tau_i)$, belongs to the worst-interfering type, hence generating L_{bus} delay on τ_i. However, corunner tasks generate requests of different types, each of which incurs a different interference on τ_i. This model takes this into account and breaks down the number of requests of the corunners between *l2h*, *l2m*, *s2h* and *s2m*:

$$b_j = b_j^{l2h} + b_j^{l2m} + b_j^{s2h} + b_j^{s2m} \qquad (9)$$

The order of these requests, from most interfering to less interfering is, *l2h*, *l2m*, *s2h* and *s2m* (see Section IV-C).

To compute the CDB_{bus}, we pair each interfered request (those coming from τ_i) with the worst eligible interfering request available from each contending core. We start pairing the accesses with the most interfering type (*l2h*) until this interfering type is consumed. The remaining $b'_i = max(0, b_i - b_j^{l2h})$ requests from τ_i are paired with the next interfering type (*l2m*). The remaining $b''_i = max(0, b'_i - b_j^{l2m})$ with *s2h* and finally the remaining $b'''_i = max(0, b''_i - b_j^{s2h})$ with *s2m*. With this CDB_{bus} is computed as follows:

$$\begin{aligned} CDB_{bus} \;\; = \textstyle\sum_{\tau_j \in c(\tau_i)} [\;\; & \min\left(b_i, b_j^{l2h}\right) \times l_{l2h} + \\ & \min\left(b'_i, b_j^{l2m}\right) \times l_{l2m} + \\ & \min\left(b''_i, b_j^{s2h}\right) \times l_{s2h} + \\ & \min\left(b'''_i, b_j^{s2m}\right) \times l_{s2m}] \end{aligned} \qquad (10)$$

It is worth noting that the type of the requests generated by τ_i are equally affected by each type of request of its corunner. That is, the interference is determined by the type of the request of the corunner task τ_j only.

B. Memory Prediction Model

To compute CDB_{mem} we apply the same models as for the bus. As explained in Section III, there are two different types of request in the memory, read and write. We assume a task τ_i with m_i requests to the memory and contender tasks $\tau_j \in c(\tau_i)$ with $m_j = m_j^{read} + m_j^{write}$ accesses to the memory each and $m'_i = max(0, m_i - m_j^{read})$. The highest delay in memory is given by $L_{mem} = \max\left(l_{read}, l_{write}\right)$.

Under these constraints the theoretical Upper-Bound Delay model is given by:

$$CDB_{mem} = m_i \times samd = m_i \times (N_c - 1) \times L_{mem} \qquad (11)$$

The model based on single request types is as follows:

$$CDB_{mem} = \sum_{\tau_j \in c(\tau_i)} \min\left(m_i, m_j\right) \times L_{mem} \qquad (12)$$

The model based on multiple request types is as follows:

$$\begin{aligned} CDB_{mem} = \sum_{\tau_j \in c(\tau_i)} & \min\left(m_i, m_j^{read}\right) \times l_{read} + \\ & \min\left(m'_i, m_j^{write}\right) \times l_{write} \end{aligned} \qquad (13)$$

From previous discussions it follows that our model builds on two pieces of information: the latency each request uses each shared resource and the number of accesses performed by each task to each shared resource. We describe both in the following subsections.

C. Deriving Access Latencies

Bus. Our model uses as an input the time each request uses each shared resource, which correspond to the bus and memory in our reference architecture. For the bus, in the NGMP, our model requires deriving the bus usage latency of *l2h*, *l2m*, *s2h* and *s2m*[4]. Since documentation typically does not provide this information, we derived it empirically.

To do so, first we executed a benchmark performing a given type of bus operations as the Task Under Analysis (*tua*), or interfered task; against a range of other benchmarks, or corunner tasks, performing all of them the same type of accesses (which may be a different type as for the *tua*). For instance, in one experiment the *tua* performs *l2h* accesses and the corunner tasks *s2h* accesses.

As a result of performing this process we reached the following three observations:

1) The execution time of the *tua* depends on the type of accesses performed by the corunner tasks. Thus, given a *tua*, its execution time may not be the same if corunners perform *l2h*, *l2m*, *s2h* or *s2m* accesses.
2) The impact of corunner tasks on the *tua* is linear with the number of corunners. Therefore, if the execution time of the *tua* in isolation (normalized) is 1, and it grows to $1+K$ when running against one corunner, then the execution time against C corunners can be upper bounded as $1 + C \times K$ (further considerations on this matter can be found in [9]). In the particular case where corunner tasks are run in all other cores, the execution time of the *tua* is:
$$1 + (N_c - 1) \times K \tag{14}$$
3) The impact of interferences in the execution time of the *tua* is independent of the particular access type performed by the *tua*. Therefore, the execution time in isolation grows by $(N_c - 1) \times K$ when all corunners perform the same type of accesses (i.e. *l2m*), but K depends solely on the type of accesses of the corunners, not on the type of accesses of the *tua*. This can be explained because the interference on the AMBA AHB bus depends only on the arbitration time [14], which in fact depends only on the time the higher priority corunners use the bus and not on the interfered request which is requesting the bus and has to wait the same amount of time regardless its particular access type.

To infer the latencies we take as a reference the *l2h* benchmark that constantly accesses the L2 cache and hence the bus. Further since the benchmark always hits in L2, each request on the bus has a short turn-around time. This benchmark is executed as *tua* in a workload comprising 3 corunner benchmarks, which correspond to the 3 remaining cores. The corunners perform accesses of the same type to the bus continuously. Hence, there are 4 different workloads depending on the type of access performed by the other three corunners: *l2h*, *l2m*, *s2h*, *s2m*.

Figure 2 shows the measured execution time for all workloads. To infer the bus latencies, we divide the execution time overhead of the *tua* with respect to the execution time in isolation by the amount of contenders (3 in each case) and then

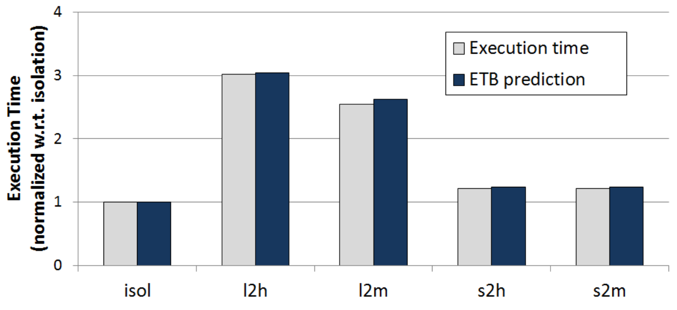


Fig. 2: Execution time and ETB of *l2h* benchmark in different workloads

divide these cycles by the amount of bus accesses performed by each contender. For instance, given an execution time of T_{isol} for the *tua* in isolation and T_{l2h} for the *tua* against 3 *l2h* corunners, the interference of an *l2h* access is obtained as follows where N_{req} is the number of *l2h* requests performed by each corunner:

$$l_{l2h} = \left\lceil \frac{T_{l2h} - T_{isol}}{(N_c - 1) \times N_{req}} \right\rceil \tag{15}$$

This way we obtain the number of interference cycles per bus access type: 9, 7, 1 and 1 for l_{l2h}, l_{l2m}, l_{s2h} and l_{s2m} respectively[4]. With these latencies we compute CDB_{bus} with Equation 6 and build the ETB prediction shown in Figure 2, which is computed using Equation 2.

Techniques to improve the confidence on derived bus latencies are proposed in [9]. Part of our future work consists of integrating those methods on top of our model and compare them against our method to derive latencies. Nevertheless, our prediction models are compatible with any method to obtain the access latencies.

Memory. The approach followed to obtain memory latencies is analogous to that for bus latencies with some small differences. First, instead of using benchmarks accessing the bus, we use *l2m* as *tua*, which performs memory reads. As corunner tasks we use first 3 copies of a *l2m* benchmark, that generates memory reads. The latency of memory reads obtained in this case is 18 cycles. In the second experiment we use 3 copies of *s2m* as corunners. The latency of memory writes obtained is again 18 cycles because there is no difference between read and write operations in terms of memory interference since, in both cases, the timing is defined by the time to open and close the memory page or row, which is identical for both.

D. Deriving Access Counts

The NGMP provides 16 PMCs that can be configured with different events and can be measured using the commercially available tool GRMON2 [4]. Among other events, we are interested in the per-core bus reads and writes (0x40 - 0x50 in [5]) and per-core L2 hits and misses (0x60 - 0x61).

Bus. The total number of L2 accesses (i.e. hits and misses) corresponds to the number of bus accesses. However, there is no way to break down L2 hits/misses into reads and writes,

[4]Please note that these latencies are not the same as those obtained in Section III-B for the L2 cache.

i.e. it is not possible to determine exactly the number of l2h, l2m, s2h and s2m accesses.

In this scenario our approach is to estimate those values in the most pessimistic way: Given task τ_j, we can obtain the number of L2 hits and misses, b_j^h and b_j^m, and the number of bus read and writes, which is equivalent to the number of L2 loads and stores, b_j^l and b_j^s. Our goal is to distribute b_j^h, b_j^m, b_j^l and b_j^s into b_j^{l2h}, b_j^{l2m}, b_j^{s2h} and b_j^{s2m} such that their total impact is maximized. For the l_{l2h}, l_{l2m}, l_{s2h} and l_{s2m} latencies in our reference architecture, the following equations maximize the impact. First, we assume the maximum amount of requests from the worst possible interfering request, i.e. l2h:

$$b_j^{l2h} = \min(b_j^l, b_j^h) \tag{16}$$

Then we subtract this value from b_j^l and b_j^h, obtaining $b_j^{\prime l} = max(0, b_j^l - b_j^{l2h})$ and $b_j^{\prime h} = \max(0, b_j^h - b_j^{l2h})$, and repeat the algorithm with the next worst interferers, i.e. l2m, s2h and then s2m, with $b_j^{\prime s} = max(0, b_j^s - b_j^{s2h})$ and $b_j^{\prime m} = \max(0, b_j^m - b_j^{l2m})$, to obtain b_j^{l2m}, b_j^{s2h} and b_j^{s2m}.

$$b_j^{l2m} = \min(b_j^{\prime l}, b_j^m) \tag{17}$$

$$b_j^{s2h} = \min(b_j^s, b_j^{\prime h}) \tag{18}$$

$$b_j^{s2m} = \min(b_j^{\prime s}, b_j^{\prime m}) \tag{19}$$

Once we have all accesses properly classified as l2h, l2m, s2h and s2h for each contending task τ_j, we can proceed with the model described before.

Memory. Our current implementation does not provide access counters for the memory controller. Hence, the exact number of memory accesses cannot be obtained, even though L2 cache misses are known, since there is no way of accounting indirect memory accesses such as writes generated by evictions of dirty L2 lines. The actual number of memory accesses can either be estimated using the number of L2 misses, which is a lower bound of the memory accesses or estimated with the number of L2 misses and the number of bus writes, which is an upper bound.

E. Assumptions

Our model is based on the assumption that the number of accesses of a task is not affected by the contenders. This happens only if the L2 is partitioned, i.e. not shared. Otherwise, the number of accesses to the bus or memory for a task executed in isolation does not match those obtained when running along with other tasks.

Also our model assumes that no timing anomalies [24] are present. Timing anomalies is an open research field, and is difficult to prove that a real processor is time anomaly free [16]. Nevertheless, if timing anomalies can occur, they cannot trigger a domino effect by construction, i.e. it is a *compositional architecture with constant-bounded effects* according to the classification in [32]. In those architectures, which may experience timing anomalies but no domino effects, the impact of timing anomalies can be accounted for easily by counting how many times they can be triggered and padding ETBs by the product of this count and the maximum impact of one timing anomaly. This approach is fully in line with our prediction model that accounts for contention in shared resources. Further, note that our model has no impact on timing anomalies, which occur (or not) regardless of our model.

V. Experimental Results

We evaluate our proposals on a real GR740 [6] FPGA prototype on a Xilinx ML510 board. We used the commercially available Cobham Gaisler GRMON2 [4] debug monitor software to directly extract the PMC from the statistic unit of the GR740, without affecting execution. The model is directly constructed from the readings obtained in one execution of each task, i.e. no further post processing is required.

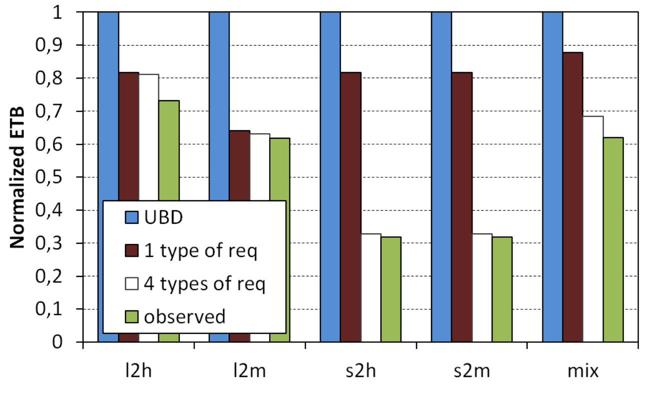
A. Bus and memory prediction models

Our first experiments put the shared resources under high pressure to test the tightness of the bounds obtained with the prediction model. To that end we use as reference applications a set of synthetic kernels [12] that inject constant high pressure either on the shared bus or on the shared memory. The Bus-Stressing Kernel, or *bsk*, comprises memory read and write requests that always miss the L1 and hit the L2, thus maximizing the traffic on the bus. This is done by having 5 memory accesses that access the same set of the L1 cache, thus exceeding its 4 ways. The same approach is used for the Memory-Stressing Kernel (*msk*) that comprises memory read requests that always miss on the L1 and also miss on the L2.

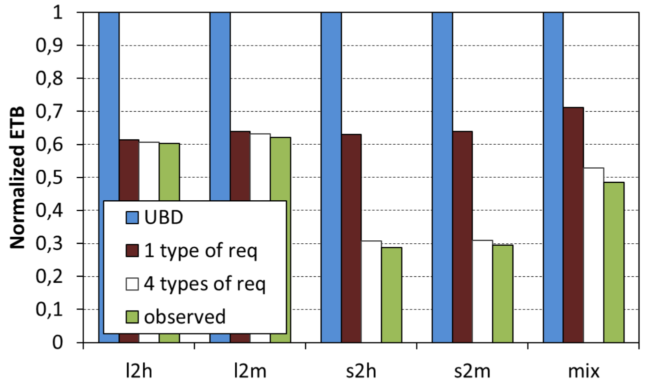
In all experiments we use one reference task (*tua*) and three tasks as corunners. In particular, as reference task on which ETB is to be derived we use bsk-ld-40% in which 40% of its instructions are loads that access the bus. As corunner tasks we use *bsk* whose frequency of access is 40% and 5%, i.e. 40% and 5% of the instructions are accesses to the bus. Those *bsk* used as corunners access the bus with requests of a different type across experiments: *l2h*, *l2m*, *s2h*, *s2m* and a *mix*, which consist of a *l2h*, a *l2m* and a *s2h bsk* together.

Figure 3a and Figure 3b show the result for bsk-ld-40% when the frequency of access of the contenders is 40% and 5% respectively. In both figures we show the ETB when using the UBD approach [12] or our approach with a single-type and four types of requests; and the observed execution time. In all cases, the predicted ETB estimates are above the observed execution time. The UBD model, since it assumes that every access of the task under analysis suffers $samd$, leads to the highest ETBs. Our model, that accounts only for the contention the task under analysis suffers, tightens the ETB. As presented in the previous sections, if the method is made aware of the request types and their associated latency (4 types of request) ETBs are further reduced.

In Figure 3b, the corunners make fewer accesses than in Figure 3a. For the UBD approach this has no impact since it only focuses on the number of requests of the task under analysis that remains the same. Instead, our models reduce ETBs since they effectively capture the fact that the corunners make fewer accesses.



(a) contenders-40%



(b) contenders-5%

Fig. 3: ETB for each bus prediction model: UBD (fully time composable requests); And our approach with 1 and 4 request types.

We performed the same experiment for the memory model, using msk-ld-40% as *tua* and corunners with 40% and 5% of memory accesses. In this case, the single type of request or multiple types of request models are equivalent, since read and writes to memory have exactly the same impact. The results are analogous to those obtained for the bus model. Therefore, we omitted the figures since they provide no further insights.

B. EEMBCs

As final evaluation we apply the whole prediction model with the EEMBC Autobench suite [20] as reference applications. We run each EEMBC benchmark under a relatively high pressure scenario composed of two tasks, one continuously accessing the bus (*bsk*) and the second accessing the memory (*msk*). In this scenario, neither the bus nor the memory controller suffer the highest pressure, since that requires all remaining cores accessing simultaneously each resource [22].

No memory accesses. As presented in Section IV-D, there is not a specific PMC to measure the number of accesses to the memory. In order to cancel out the impact of this, in a first experiment we focus on the case in which the *tua* is run twice in a row and measurements are taken during the second run. Due to the small footprint of EEMBC Autobench, this results in almost zero misses in the second run.

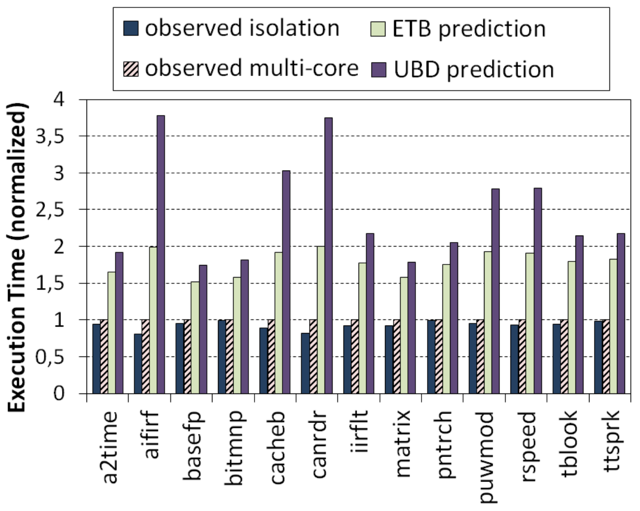


Fig. 4: ETBs for EEMBC when assuming a no cache misses.

Under that assumption, we present the results of our model and UBD in Figure 4. Recall that in each workload we run one EEMBC benchmark executed and the two contenders presented above. Results are normalized w.r.t. the execution time of the EEMBC in the workload. We show the execution time in isolation, the execution time in the workload and the predicted ETB using the multiple type of requests prediction model for the bus and memory, as well as the UBD. We can clearly see that our prediction model reduces the pessimism of the UBD model by 67% being only 79% higher than the actual execution time. In Figure 3, the observed execution time is much closer to the prediction when compared with Figure 4. This is because the scenario in Figure 3 is designed to experience severe contention, whereas the scenario described here experiences much lower contention (far below the upper-bounded contention).

General case. Our next step is to evaluate the natural case in which programs perform memory accesses. According to Section IV-D, we can estimate the access to the memory using the number of L2 misses. The number of L2 misses does not consider the dirty evictions that generate memory accesses. To take into account the dirty evictions into the memory accesses we can use either an optimistic lower bound based on the number of L2 misses or a pessimistic upper bound based on the number of L2 misses plus the bus writes. To that end, we build three scenarios:

- Pessimistic scenario. We assume that every write operation results in a dirty eviction, i.e. an access to memory.
- Accurate scenario. In this case, from a simulation tool we derive the exact number of memory accesses.
- Optimistic scenario. We disregard the dirty evictions and take the number of L2 misses as memory accesses.

Figure 5 shows the results obtained with our model under each of the previous scenarios and the observed execution time in the workload and isolation. These results provide a good estimate of the benefits of improving our reference design with a PMC that explicitly measures memory accesses.

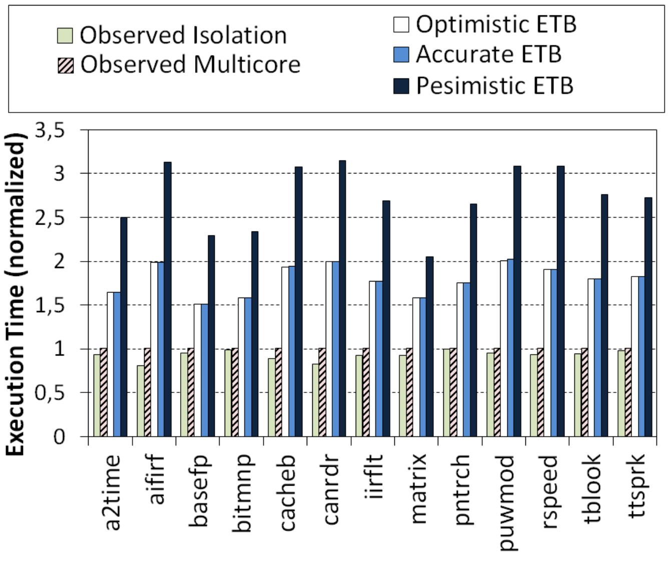


Fig. 5: ETBs for EEMBC under the optimistic, pessimistic and accurate scenarios.

As expected the pessimistic scenario that considers all writes as dirty evictions is overly pessimistic. In particular it is 138% more pessimistic than the actual observed execution time. The accurate scenario in which we assume that the PMC for access count exists leads to very tight estimates, 64% more pessimistic than the actual observed execution time and less than 0.1% more pessimistic than the optimistic scenario. This is due to the small memory footprint of the EEMBC benchmark, that fit on the L2 cache. As a result, the number of dirty evictions is close to zero in most scenarios.

VI. Conclusions

In this paper we present a prediction model of the shared resource contention for the GR740 that takes into account the number of accesses and their type for a given task and its corunner tasks, which can be easily obtained with PMCs. The model abstracts (i.e. makes worst-case provisions) for the way in which requests interleave in time, which would challenge time composability since such time interleaving could easily change during operation.

Derived Execution Time Bounds (ETBs) are shown to be accurate and tighter than fully-time composable ETBs. Those derived estimates are valid for any workload in which the task runs as long as the number of accesses (per type) is smaller than those assumed at analysis. This provides a good balance between tightness and time composability.

VII. Acknowledgements

The research leading to this work has received funding from: the European Space Agency under contracts 4000109680, 4000110157 and NPI 4000102880. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

References

[1] B. Akesson et al. Predator: a predictable SDRAM memory controller. In *CODES+ISSS*, 2007.
[2] F. J. Cazorla et al. Multicore OS benchmarks. Technical report, European Space Agency, 2012.
[3] S. Chattopadhyay et al. A unified WCET analysis framework for multicore platforms. *ACM Trans. Embed. Comput. Syst.*, 13(4), Apr. 2014.
[4] Cobham Gaisler. *GRMON2-User Manual Version 2.0.62, March 2015*.
[5] Cobham Gaisler. *NGMP Preliminary Datasheet Version 2.1, May 2013*.
[6] Cobham Gaisler. *Quad Core LEON4 SPARC V8 Processor - GR740-UM-DS-D1 - Data Sheet and Users Manual, 2015*.
[7] Cobham Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-N2X Data Sheet and Users Manual Version 2.1*, 2015.
[8] G. Fernandez et al. Contention in multicore hardware shared. resources: Understanding of the state of the art. In *WCET Workshop*, 2014.
[9] G. Fernandez et al. Increasing confidence on measurement-based contention bounds for real-time round-robin buses. In *DAC*, 2015.
[10] G. Fernandez et al. Resource usage templates and signatures for COTS multicore processors. In *DAC*, 2015.
[11] G. Fernandez et al. Seeking timecomposable partitions of tasks for cots multicore processors. In *ISORC*, 2015.
[12] M. Fernandez et al. Assessing the suitability of the NGMP multi-core processor in the space domain. In *EMSOFT*, 2012.
[13] J. Jalle et al. Deconstructing bus access control policies for real-time multicores. In *SIES*, 2013.
[14] J. Jalle et al. AHRB: A high-performance time-composable amba ahb bus. In *RTAS*, 2014.
[15] H. Kim et al. Bounding memory interference delay in cots-based multi-core systems. In *RTAS*, 2014.
[16] T. Lundqvist et al. Timing anomalies in dynamically scheduled micro-processors. In *Real-Time Systems Symposium*, 1999.
[17] J. Nowotsch et al. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *ECRTS*, 2014.
[18] M. Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.
[19] M. Paolieri et al. Timing effects of DDR memory systems in hard real-time multicore architectures: Issues and solutions. *ACM Trans. Embed. Comput. Syst.*, 12(1s), 2013.
[20] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[21] P. Puschner et al. Towards composable timing for real-time software. In *1st International Workshop on Software Technologies for Future Dependable Distributed Systems*, 2009.
[22] P. Radojković et al. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *ACM Trans. Archit. Code Optim.*, 2012.
[23] Rapita systems Ltd. Rapita Verification Suite. http://www.rapitasystems.com/products/rvs.
[24] J. Reineke et al. A definition and classification of timing anomalies. In *WCET*, 2006.
[25] J. Rosen et al. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS*, 2007.
[26] RTCA & EUROCAE. DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations.
[27] A. Schranzhofer et al. Timing analysis for resource access interference on adaptive resource arbiters. In *RTAS*. IEEE, Apr 2011.
[28] H. Shah et al. Measurement based WCET analysis for multi-core architectures. In *RTNS*, 2014.
[29] M. Slijepcevic et al. Time-analysable non-partitioned shared caches for real-time multicore systems. In *DAC*, 2014.
[30] A. West. NASA Study on Flight Software Complexity. Final Report. Technical report, NASA, 2009.
[31] R. Wilhelm et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.
[32] R. Wilhelm et al. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(7):966–978, July 2009.

Predictable composition of memory accesses on many-core processors

Quentin Perret*†, Pascal Maurère*, Éric Noulard†, Claire Pagetti†, Pascal Sainrat‡, Benoît Triquet*
*Airbus Operations SAS, Toulouse, France. firstname.lastname@airbus.com
†ONERA, Toulouse, France. firstname.lastname@onera.fr
‡University of Toulouse, France. sainrat@irit.fr

***Abstract*—The use of many-core COTS processors in safety critical embedded systems is a challenging research topic. The predictable execution of several applications on those processors is not possible without a precise analysis and mitigation of the possible sources of interference. In this paper, we identify the external DDR-SDRAM and the Network on Chip to be the main bottlenecks for both average performance and predictability in such platforms. As DDR-SDRAM memories are intrinsically stateful, the naive calculation of the Worst-Case Execution Times (WCETs) of tasks involves a significantly pessimistic upper-bounding of the memory access latencies. Moreover, the worst-case end-to-end delays of wormhole switched networks cannot be bounded without strong assumptions on the system model because of the possibility of deadlock. We provide an analysis of each potential source of interference and we give recommendations in order to build viable execution models enabling efficient composable computation of worst-case end-to-end memory access latencies compared to the naive worst-case-everywhere approach.**

***Keywords*—many-core processor, real-time, composition rules, execution model, DDR-SDRAM, Network on Chip**

I. Introduction

The increasing complexity of modern COTS processors and especially the change of architectural paradigm coming with the emergence of many-core processors involve new challenges to bound the worst-case execution time of real-time applications. Indeed, many-core processors aim at solving the scalability issue of multi-core processors by changing the inter-core communications methods from implicit shared-memory mechanisms to explicit point-to-point communications through one or several Network on Chip (or NoC) and by allocating private on-die memory areas to each core or group of cores. In the frame of real-time systems, this new architectural paradigm also brings new challenging research topics.

The important multiplication of cores implies that the external memory will also be shared much more. Moreover, a transaction with the external memory initiated by a core will now have to go through a NoC, implying new potential sources of interferences. Thus, the problem of bounding the execution time of applications, and especially, the subsequent problem of bounding the memory access latencies will become increasingly hard. Moreover, the utilization of many-core processors to execute several safety critical applications will only be possible in the industry if the requirements related to incremental certification can be met. Such requirements include the need of composability to ensure decoupled certification processes of the applications.

In this paper, we propose to identify each shared resource on the memory access path and to build a composition rule describing its behaviour in the case of concurrent accesses. We show that a worst-case-everywhere approach is not viable as it implies a potentially large under-utilization of the resources due to pessimism in the calculation. The latencies of the NoC and the DDR-SDRAM appear to be particularly difficult or even impossible to bound tightly without assumptions on the potential competitors. So, we give recommendations for building viable *execution models* (ie. a set of restricting rules that must be met by the applications) in order to ease the tight calculation of Worst-Case Execution Times (or WCETs) with minimal assumptions on the behaviour of the applications.

The rest of the paper is organized as follows. Section II provides the description of the many-core platform we will consider. We identify in Section III each potential interference source on the memory transactions paths and we define all their composition rules. Section IV discusses the required background knowledge on DRAM and classical memory arbitration techniques. We evaluate the end-to-end latency a memory transaction in Section V and we provide recommendations for execution model design in Section VI. Related work is addressed in Section VII and Section VIII concludes the paper.

II. Platform description

Our platform model assumes a Commercial-Off-The-Shelf (or COTS) many-core processor (as shown in figure 1) organized in *tiles* of two different categories:

- The *Compute Tiles* have for main purpose to execute user code. They are composed of N_c^c (usually ≥ 1) computing cores, *local* memory (usually SRAM) shared by all cores inside the tile and N_{dma}^c (usually $= 1$ in Compute Tiles) Direct Memory Access (or DMA) devices to enable inter-tile communication through a Network-on-Chip.
- The *I/O tiles* are used for communication with out-of-chip components such as DDR3-SDRAM. They include N_c^{io} (usually ≥ 1) computing cores, N_{dma}^{io} (usually ≥ 1) DMAs and N_{phy}^{io} physical interfaces linked with out-of-chip components.

The tiles communicate through a Network-on-Chip (or NoC) based upon a packet-switching strategy (e.g. *wormhole switching* or *store and forward*). This implies that large

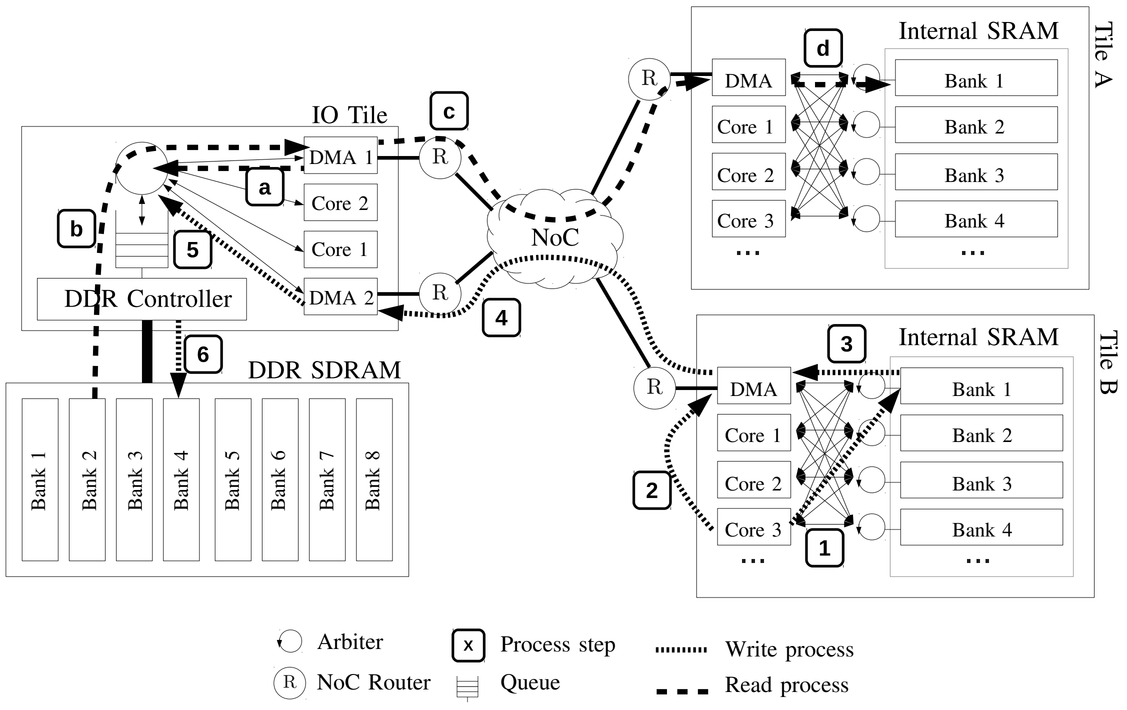


Fig. 1: Model of a many-core processor with memory accesses from computing tiles

communications over the NoC are split into packets composed of several flow control digits (or flits). In the following sections we will refer to a series of packets composing a single memory transaction as a *flow*. In this architecture, Compute Tiles are not able to issue commands directly to external RAM. The only way for Compute Tiles to interact with the main memory is to use the IO tiles as an interface to which every request must be sent explicitely by software. We explain the processes of reads and writes from/to the external memory with two examples.

Example 1 (Write process): In this example, the Core 3 of Tile B requires to write data in the bank 4 of the external DDR3-SDRAM memory. We detail each step of this write process as shown in figure 1 (the numbering is equivalent to the one of the path of the write process in figure 1):

1) *The requesting computing core (Core 3 of Tile B) writes the data to be sent in the local memory of the computing tile. As the banks of the local SRAM are shared among many potential requestors, there may be an arbitration at this level in the case of concurrent accesses to the bank.*
2) *The Core signals to the local DMA its itention to send the data.*
3) *DMA reads the data (written by Core 3 at step 1) from the local memory. Once again, any concurrent access to the same bank will involve an arbitration.*
4) *DMA sends the data through the NoC. If the amount of data to send is important, it will be split in several packets constituing a flow. All the packets will cross the NoC following the same path. If one or several parts of this NoC path are shared with other NoC flows, the arbitration between the flows will occur at packet level.*
5) *The sink DMA (DMA 2 of IO Tile) receives the packets and initiates DDR3-SDRAM write transactions. If other masters (IO Tile Cores, other IO Tile DMAs, . . .) access the external memory concurrently, an arbitration process will occur. This phase assumes that the sink DMA has been configured before reception to associate one of its reception queues to a specific DDR3-SDRAM address (an address in bank 4 of the DDR3-SDRAM here).*
6) *Once the sink DMA write(s) request(s) is/are elected by the memory arbiter, data is written into the DDR3-SDRAM array.*

Example 2 (Read process): In this example, the Core 2 of Tile A needs to read data from the bank 2 of the DDR-SDRAM to store it in the bank 1 of its local memory. We detail each step of this read process as shown in figure 1 (the numbering is equivalent to the one of the path of the write process in figure 1):

a) *The DMA of the IO Tile initiates a DDR3-SDRAM read transaction. Once again, any concurrent access to the memory with any other master will involve arbitration.*
b) *Once the DMA command is elected, data is transfered from DDR3-SDRAM to the DMA.*
c) *DMA sends packets through the NoC. Again, important amounts of data are packetized and arbitrated with concurrent flows at packet level.*
d) *DMA of the Compute Tile receives the packets and attempts to write them back into the local memory. Again, we assume that this DMA has been pre-configured to associate one of its reception queue to a specific memory area of the local memory (the bank 1 in this example).*

We remark that a read process is fairly equivalent to a write process. Indeed, a read by a computing core is equivalent to a write from an IO Tile. The difference is that the destination of the data is not the external memory but the internal memory of a Compute Tile.

In this example, the phase a) of the read process is initiated by the DMA of the IO Tile. We assume that the DMA was notified by one of the IO Tile's core that received a command from one compute tile or that has been pre-configured.

This model is representative of a certain class of tiled many-core processors such as the KALRAY MPPA®-256 [1]. In the next sections, we try to estimate the temporal bounds of any individual access on the identified sources of interference with no assumption on the behaviour of other potential requesters.

III. SOURCES OF INTERFERENCE

In this section, we will refer as a memory *transaction* to the high-level application demands of memory. Each transaction can be composed of several memory *requests* at external memory controller level.

Definition 1 (Worst-Case-Everywhere Approach): We denote as a Worst-Case-Everywhere Approach *a method for bounding the memory access time of an individual requester with no assumptions on the competitors on each shared resource. In this context, one must consider only the worst-case behaviour of the competitors to provide a safe bound.*

A. Local memory arbitration

For simplification purpose, the local memory of the Compute Tiles is assumed to be Static Random Access Memory (or SRAM) for which there is a simple access protocol and no refresh is required. The local memory of each computing tile is split into N^c_{bank} banks. The memory frequency is f^c_{mem} and the data bus is w^c_{mem} bytes large. There are $N^c_c + N^c_{dma}$ potential memory requesters in a compute tile. We assume each requester to own a private access path to the memory. Concurrent accesses to different banks have no impact on bandwidth. Concurrent accesses to a single bank are arbitrated with a Round-Robin policy. So, for a memory transaction of s_{trans} bytes, the total duration is:

$$t_{SRAM}(N_{req}, s_{trans}) = \left\lceil \frac{s_{trans}}{w^c_{mem}} \right\rceil \times \frac{N_{req}}{f^c_{mem}} \qquad (1)$$

In a worst-case-everywhere approach, one must always consider $N_{req} = N^{max}_{req} = N^c_c + N^c_{dma}$. So, we can estimate the worst case latency of a local memory transaction by $t^{max}_{trans} = t_{SRAM}(N^{max}_{req}, s_{trans})$.

B. Network on Chip

In this section, we assume a NoC designed upon a wormhole switching strategy. The access to the NoC is enforced by the DMA. Communications upon the NoC are split into packets having a maximum size of s^{max}_{pk} flits of payload where the size of one flit is s_{flit} bytes. The number of non-payload flits by packet is s_{header}. The maximum frequency at which flits can transit on the NoC is f_{NoC}. We show in figure 2 the model of a NoC router. A router R_i is composed five interfaces named East, West, North, South and Local. The Local interface is not represented in figure 2 for clarity. The arbiter at each interface implements a Round Robin policy at packet level. The arbiters are work conserving, meaning they are never idle when there is a packet to send. Consequently, they do not introduce undesired gaps between packets.

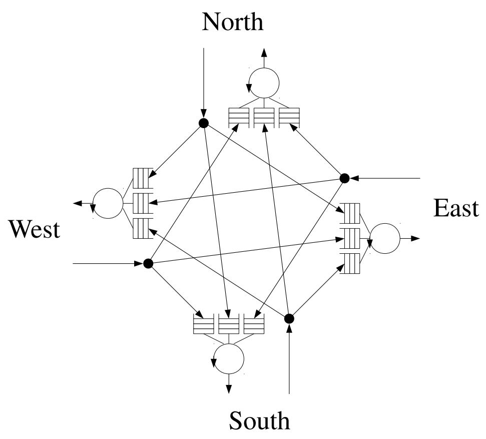


Fig. 2: Model of a NoC router

In wormhole switched networks, one message can be holding one resource while requesting others, and thus, cause a deadlock [2]. We show an example of deadlock in figure 3.

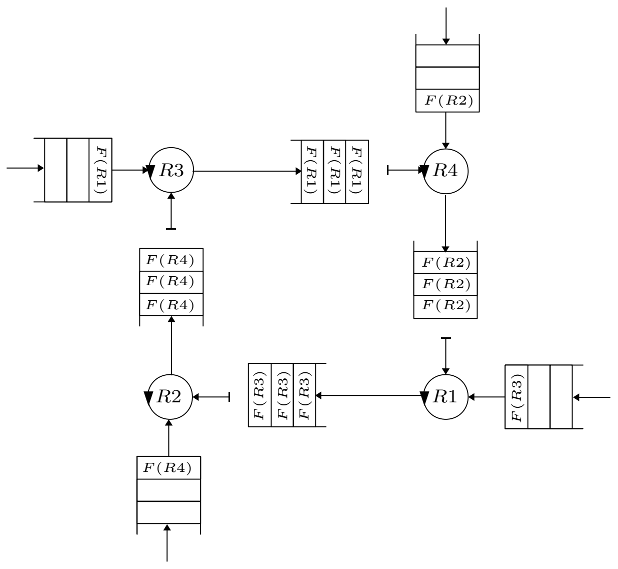


Fig. 3: Deadlock on a wormhole routed NoC

In this example we can see the flits $F(Rx)$ of 4 packets in the FIFO queues of the interfaces of 4 routers. 3 out of 4 flits $F(R1)$ at destination of the router $R1$ went through the router $R3$ and are stored in one queue of the router $R4$ waiting for availability of the link to $R1$. Because of the back-pressure mechanism, the fourth $F(R1)$ flit is still queueing in $R3$ as the queue of $R4$ is full. It is also maintaining occupied the link between $R3$ and $R4$ as all the flits of one packet must be consecutive. At the same time, the flits $F(R2)$ blocking the flits $F(R1)$ are waiting for the link between $R1$ and $R2$ to become idle. Similarly, the flits $F(R3)$ occupying this link are waiting for the link between $R2$ and $R3$ to become idle but this link is occupied by the $F(R4)$ flits themselves waiting for the $R3$ to $R4$ link that is occupied by the $F(R1)$ flits. We can see clearly here the occurrence of an unsolvable cyclic dependency leading to a deadlock. Such a problem can happen if no assumptions are made on the software accessing the NoC. So, a worst-case everywhere approach is not applicable. In the literature, the attempts to bound the end-to-end delays of wormhole switched networks usually assume specific routing algorithms [3] or acyclic *Channel Dependency Graphs* [2] or regulation of traffic injection to ensure deadlock-free executions. For example, in [4], the authors present an approach ensuring no overflow of the KALRAY MPPA®-256's NoC routers FIFOs using the hardware limiters properly configured with Network Calculus [5]. Thanks to the design of the NoC routers and the FIFO overflow avoidance, no deadlock can happen. However, in this case, the effective latency of any NoC packet depends on the contribution of other participant and thus does not provide *composability* (even if the maximum end-to-end latency can indeed be bounded). An other possible approach that offers composability is to compute an off-line TDMA scheduling of the NoC in order to provide periodic time windows to each task during which they access the NoC with no concurrents [6]. We argue anyway that static hardware-based routing policy offers less flexibility than explicit routing decided by software (at the cost of an overhead implied by the route planning obviously). This flexibility enables to choose complex routes that may help the system designers to avoid route conflicts when trying to compute efficient TDMA scheduling tables or to build acyclic Channel Dependency Graphs.

C. Main memory access

We consider a DDR3-SDRAM memory as defined by the JEDEC standard [7]. As shown in figure 1, we assume that concurrent accesses to the memory are arbitrated before being issued to the controller. So, the problem of bounding the memory latencies can be divided into two subproblems:

1) what is the policy used by the controller to serialize several parallel accesses ?
2) how does the memory react to a certain sequence of requests ?

As the detailed explanation of both problems comes with prerequisites, we discuss them in section IV-C after an introduction on DRAM technology.

IV. DRAM BACKGROUND

We present the basics of DRAM in order to explain the inherent timing constraints related to this technology and we address the problem of concurrent memory accesses. More detailed information about DRAM are available in [8].

A. DRAM technology

The Dynamic Random Access Memory (DRAM) is a simple, cheap and compact type of memory widely used in modern computers. A DRAM device is usually composed of several DRAM *banks*. A bank is an independent array of DRAM *cells* where each cell stores 1 bit of data. A cell is composed of a capacitor and a transistor able to connect the storage capacitor to the *sense amplifiers* of the bank. The sense amplifiers are acting as an interface between the cells *rows* and the memory controller. The sense amplifiers of one bank can be connected to only one row at a time. We will refer to the currently connected row as the *active* row or the *opened* row. Any *column access command* (ie. read or write) must be issued to the opened row. To issue requests on closed rows, the opened row must be *precharged* (or *closed*) first so that the according row can be activated.

B. Bank commands

We identify five main bank commands (ACT, PRE, RD, WR, REF). We detail each of them in the following sections.

1) Row activate: The purpose of a Row Activate command (or ACT) is to connect one row in the bank to the sense amplifiers. The important timing parameters related to the ACT command are:

- t_{RCD}: Row to Column Delay. The time the memory controller must wait after the ACT before it can issue a Column Read or Write command.
- t_{RAS}: Row Access Strobe. Minimum time a row must remain opened before the next precharge.
- t_{RRD}: Row activate to Row activate Delay. Minimum time required between two ACT commands.
- t_{FAW}: Four row Activation Window. Sliding window during which no more than 4 ACT commands can be issued.

2) Read: A Column Read command (or RD) is issued on an opened row in order to transfer data from the DRAM array to the memory controller. In modern DDR3-SDRAM, data is moved in relatively small bursts. We note the size in bytes of one burst s_{burst}. The important timings related to the RD command are:

- t_{CAS}: Column Access Strobe. Duration required by the memory to place on the data bus the requested data. This parameter is also often noted t_{CL}.
- t_{burst}: The time (in cycles) required to transfer a complete burst. If the memory data bus is w_{bus} bytes large, a complete burst will be transfered in s_{burst}/w_{bus} beats of data. In DDR3-SDRAM systems, the double data rate mechanism allows to transfer two beats of data by cycle. So, $t_{burst} = s_{burst}/(2 \times w_{bus})$ cycles.

3) Write: A Column Write command (or WR) is issued on an opened row in order to transfer data bursts from the memory controller to the DRAM array. The important timings related to the WR command are:

- t_{burst}: Same as RD bursts.

- t_{CWD}: Column Write Delay. Delay between the WR command and the placement of data on the bus.
- t_{WTR}: Write To Read delay. Minimum time between a WR and a RD command. This constraint is related to the bus switching time. t_{WTR} is not local to a bank but a global device constraint.
- t_{WR}: Write Recovery delay. Minimum amount of time to wait after a column write command before a precharge command can be issued.

4) Precharge: A precharge command (or PRE) has for main purpose to disconnect the current row. The important timings related to the PRE command are:

- t_{RP}: Row Precharge delay. The time required to disconnect the opened row from the sense amplifiers.
- t_{RC}: Row Cycle. $t_{RC} = t_{RAS} + t_{RP}$ is a commonly used indicator for DDR3-SDRAM performance.

5) Refresh: Refresh commands must be issued periodically to all the DRAM rows in order to avoid data corruption. We assume that the memory controller uses a simple Auto-refresh policy. In this case, a REF command operates in parallel in all banks and refreshes one or several rows in each bank. The important timings related to the REF command are:

- t_{REFI}: Refresh interval. Time interval between two REF commands issued by the controller.
- t_{RFC}: Refresh Cycle. Duration of one refresh cycle.

To safely upper-bound the latency of a sequence of memory access, the penalties related to the REF commands must be taken into account. In [9], the authors provide a method to take calculate these penalties with equation 2.

$$t_{seq}^{ref} = t_{seq} + \left\lceil \frac{t_{access}}{t_{REFI} - t_{RFC}} \right\rceil \times t_{RFC} \tag{2}$$

Where t_{seq} is the latency of the sequence of memory accesses calculated without taking into account refreshes. $\left\lceil \frac{t_{seq}}{t_{REFI} - t_{RFC}} \right\rceil$ gives the maximum number of refreshes that may occur during t_{seq}. Therefore, the refresh cycle time t_{RFC} is added to t_{seq} as many time as it is possible in the worst case. As the refresh-related penalties can be calculated separately from the calculation of t_{seq} using equation 2, we do not take them into account in the rest of the paper.

In order to give to the reader the order of magnitude of each previously enumerated timing parameter, we provide in table I the values extracted from the technical documentation of a Micron DDR3L SDRAM module [10] composed of 8 banks of 512MiB each.

C. Concurrent accesses

In order to analyse the memory behaviour when accessed by several competitors, we decompose the analysis in two steps. At first, we examine the response of a memory bank to a specific sequence of commands and then we identity the arbitration mechanisms between the competitors.

Parameter	Nanoseconds	Cycles	Data beats
t_{CK}	1.25	1	2
t_{BURST}	5	4	8
t_{CAS}	13.75	11	22
t_{RP}	13.75	11	22
t_{RCD}	13.75	11	22
t_{WR}	21.25	17	34
t_{WTR}	7.5	6	12
t_{RAS}	35	28	56
t_{RC}	48.75	39	78
t_{FAW}	30	24	48
t_{RRD}	6.25	5	10
t_{CWD}	10	8	16
t_{RFC}	260	208	416
t_{REFI}	3906	3125	6250

TABLE I: Timing parameters of Micron module [10]

Prev. cmd.	Cur. cmd	Timing parameter
RD	RD	t_{burst}
RD	WR	$t_{CWD} + t_{burst}$
RD	PRE	$t_{RC} - t_{read}^{max}$
WR	RD	$t_{CAS} + t_{burst} + t_{WTR}$
WR	WR	t_{burst}
WR	PRE	$max(t_{WR}, t_{RAS} - t_{write}^{max}) + t_{RP}$
ACT	RD	$t_{CAS} + t_{burst}$
ACT	WR	$t_{CWD} + t_{burst}$
ACT	PRE	t_{RC}
X	ACT	t_{RCD}

TABLE II: Visible timings of commands at bank level

1) Bank level: At bank level, we can see as input a series of low level commands (ACT, PRE, RD, WR) on one bank and as output the resulting time required to complete the whole sequence of commands. We detail in table II the *visible* timing of each command depending on the previous command issued to the same bank. So, the time needed by one command sequence can be calculated by summing the parameters of table II corresponding to each command.

Example 3 (Calculation of the duration of 4 commands sequence on one bank): As shown in figure 4, we consider a sequence of four commands (one ACT followed by 3 RD). The three first commands are issued back to back and the last one is issued after a gap of 3 cycles. With the parameters of table I and the expressions of table II we calculate the time required to complete the whole sequence $t_{seq} = 37$ *cycles with:*

$$t_{seq} = \underbrace{t_{RCD}}_{ACT} + \underbrace{t_{CAS} + t_{BURST}}_{1^{st}\ RD} + \underbrace{t_{BURST}}_{2^{nd}\ RD} + \underbrace{t_{BURST}}_{3^{rd}\ RD} + t_{GAP}$$

2) Controller level: The arbitration strategy implemented in order to serialize several concurrent memory transactions varies from one controller to another. One of the most widely used arbitration policy in COTS controllers is the First-Ready First-Come First-Serve (or FR-FCFS). With FR-FCFS, requests on already opened DRAM rows are issued first, and once no pending request targets an opened row, the oldest request goes first. Bounding the memory access time of a FR-FCFS-based controller can be challenging since an aggressive implementation of this arbitration policy can imply starvation as new requests are likely to be issued before older ones.

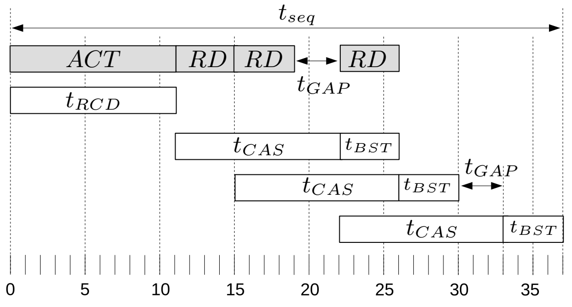


Fig. 4: Sequence issued to a bank with $t_{BST} = t_{BURST}$

The real implementation of the FR-FCFS policy often slightly differ from one COTS controller to another. For instance, the differences can be related to RD and WR grouping, to starvation avoidance (some controllers have a *cap* [11] for example) or to the impact of DRAM refreshes on priorities. For this reason, the accurate modeling of the arbitration policy of COTS controllers is target dependent. In the following section, we propose an example of modeling with the KALRAY MPPA®-256's arbiter in order to quantify its worst-case memory access time. Based on this, we will provide recommendations (that can still reasonably be applied on different COTS controllers since they do not require target-specific configurations) to reduce the pessimism implied by the worst-case-everywhere approach at the controller level.

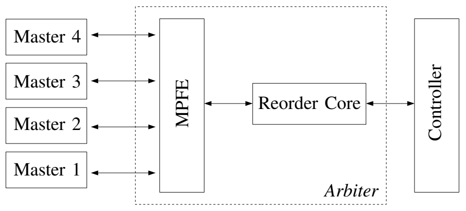


Fig. 5: KALRAY MPPA®-256's arbiter

3) KALRAY MPPA®-256's arbitration policy: As shown in figure 5, the KALRAY MPPA®-256's memory arbiter is composed of two elements. The *Multi port front end* (or *MPFE*) has several *ports*, each of which is connected to one master (DMAs Rx, DMAs Tx, IO cores, ...), and one connection to the *Reorder Core* (or *RC*). The purpose of the MPFE is to forward the requests pending on its ports to the RC one after the other. To do so, each port is assigned a priority and the highest is forwarded first. When several ports have the same priority, they are arbitrated in Round-Robin. The starvation on low-priority ports can be avoided thanks to a *starvation counter* (or *SC*). When a request arrives on a port, its SC starts decounting from a predefined value. When the SC reaches 0, the port gets the highest priority.

In order to simplify the modelling, in the rest of the paper, we will assume equal priorities on all ports and a disabled SC mechanism so that the MPFE forwards the requests in a pure Round-Robin fashion. This configuration is realistic since it can be applied on the real hardware.

The Reorder Core receives the requests forwarded by the MPFE and issue them efficiently to the controller. The RC has a queue of 8 elements that is arbitrated as follows:

1) High priority requests (same priority as for the MPFE) goes first;
2) Requests on active banks goes first;
3) Requests targeting a recently opened pages wait t_{RC} before being issued;
4) RD request goes before a WR if the previous request was a RD (same thing for WR).

Every time a request is issued to the controller, the RC accepts a new entering request from the MPFE and the 4 rules are re-evaluated.

Example 4 (Reorder Core): The following requests are present in the reordering pool:

R1: RD of priority 7 to a new page in bank 0;
R2: WR of priority 4 to an opened page in bank 1;
R3: RD of priority 4 to a new page in bank 0;
R4: RD of priority 4 to a new page in bank 1;
R5: WR of priority 7 to an opened page in bank 2;
R6: WR of priority 7 to a new page in bank 1;
R7: WR of priority 4 to an opened page in bank 3;

The requests will be served in the following order by the RC:

1) *R5: wins rule 1) with R1 and R6 and wins rule 2)*
2) *R6: wins rule 1) with R1 and wins rule 4)*
3) *R1: wins rule 1)*
4) *R7: wins rule 2) (page of R2 has been closed by R6)*
5) *R3: wins rule 3) (bank 0 is the least recently opened)*
6) *R4: wins rule 4)*
7) *R2: last request*

D. Bounding the duration of a DDR3-SDRAM transaction

In this section, we try to bound the duration of a reference memory transaction denoted τ and composed of N^{τ}_{req} requests initiated by one master and we consider a total number of N_{trans} competitors (all masters with pending memory requests including the one issuing τ). If all the possible masters are issuing memory requests simultaneously, $N_{trans} = N^{max}_{trans}$. With the previous assumptions,in the worst case, the number of arbitration round required for all N^{τ}_{req} requests to cross the MPFE is bounded by:

$$N^{max}_{round} = N^{\tau}_{req} \times N^{max}_{trans} \quad (3)$$

In the worst case, a request stays in the reorder queue at most while $2n-1$ (n is the number of element in the queue, 8 for the MPPA®-256) other requests are issued before. So, if $N^{max}_{trans} \leq (2n-1)$, several requests of τ can be located in the reordering pool simultaneously and can be issued fastly to the controller as they certainly target the same page. Otherwise, each request of τ is ensured to get out of the reordering pool before the arrival of any other request of τ. In both cases, the duration of τ is mostly dictated by equation 3. So, the maximum duration of τ can be bounded by:

$$t^{max}_{\tau} \leq (N^{max}_{round} + 2n - 1) \times t^{max}_{req} \quad (4)$$

with t_{req}^{max} the worst case request time (a read following a write with row conflict).

In the following sections, we provide numerical examples of all the previously enumerated composition rules for each identified potential source of interference and we put in evidence the part of pessimism that can be avoided by restricting the execution of the applications with a number rules.

V. Cost of composability

In this section, we explain the methodology enabling to bound the end-to-end latency of the write process of Example 1 of Section II as shown in figure 6. At first, we provide the analytical study of this example and we then provide some numerical applications in order to emphasize the pessimism implied by the worst-case-everywhere approach.

A. End-to-end latency

1) Local memory: During the phase 1, the time required by Core 3 to write the data to be sent into the Bank 1 of the Tile's local memory can be calculated with equation 1:

$$t_1(N_{req}, s_{trans}) = \left\lceil \frac{s_{trans}}{w_{mem}^c} \right\rceil \times \frac{N_{req}}{f_{mem}^c}$$

with N_{req} being the number of requesters accessing the same bank of the Computing Tile's local memory. We assume that during the phase 2, the time required by Core 3 to signal its intention to send data to the DMA is one clock cycle $t_2 = 1$.

2) Network on Chip: The phases 3, 4, 5 and 6 must be considered simultaneously as they are all impacted by the NoC management. As explained in section III-B, it is impossible to bound the NoC crossing time of a packet in complete isolation without strong assumptions on the concurrent NoC users or on the execution model orchestrating the applications. To deal with this problem, we assume that inter-application NoC traffic isolation is ensured with a pre-computed TDMA scheduling table. The respect of the TDMA requirements are ensured by trusted software granting or delaying the NoC access to the applications. Such model enables us to consider that packets may be temporary restrained at emission but those travelling in the NoC are never blocked by any concurrent at router level. So a transaction of s_{trans} bytes will require a flow ϕ_k of $N_{pk}^{\phi_k}$ packets to be completely sent:

$$N_{pk}^{\phi_k}(s_{trans}) = \left\lceil \frac{s_{trans}}{s_{pk}^{max} \times s_{flit}} \right\rceil$$

We consider that ϕ_k has been allocated a path of $N_R^{\phi_k}$ routers during a time window of L_{ϕ_k} NoC cycles every T_{ϕ_k} cycles. We assume that the length of L_{ϕ_k} is long enough for the emission of at least one packet of maximum size. As shown in figure 6, we note Δ the time between the end of the phase 1 and the emission of the first flit of the first packet. Because of the TDMA allocation of the NoC, the maximum Δ occurs when the phase 1 ends exactly at the end of one L_{ϕ_k}. In this case $\Delta^{max} = T_{\phi_k} - L_{\phi_k}$.

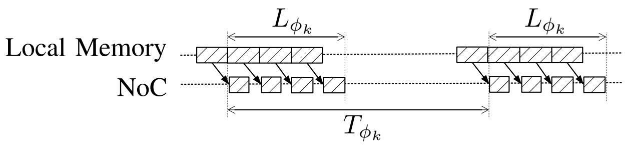


Fig. 7: Impact of N_{req} on NoC utilization

a) Consecutive packets: At first, we consider no interference at local memory level when the DMA reads the data to be injected in the NoC as shown in figure 6. In this case, the flits of all the packets are sent consecutively. We note λ_{ϕ_k} the time (in cycles) needed by one flit to cross the complete path:

$$\lambda_{\phi_k} = N_R^{\phi_k} \times (\delta_R + 1)$$

where δ_R is the latency of one router. We also assume each NoC link can be crossed by one flit each cycle. Thus the time (in cycles) needed by N_{flits} to cross the NoC is:

$$t_{NoC}(N_{flits}) = \lambda_{\phi_k} + N_{flits}$$

And, the maximum number of flits $N_{flits}^{L_{\phi_k}}$ that can be sent in one L_{ϕ_k} is :

$$N_{flits}^{L_{\phi_k}} = L_{\phi_k} - \lambda_{\phi_k}$$

So, the maximum number of packets that can be sent in one L_{ϕ_k} is:

$$N_{pk}^{NoC} = \left\lfloor \frac{N_{flits}^{L_{\phi_k}}}{s_{pk}^{max} + s_{header}} \right\rfloor \quad (5)$$

b) Non consecutive packets: As shown in figure 7, due to interference at local memory level, the DMA may not be able to read data fast enough to effectively send N_{pk}^{NoC} packets. Indeed, during a time window of L_{ϕ_k} cycles, depending on the number of local memory requesters accessing the same SRAM bank N_{req}, the maximum amount of data that can be read from the local memory can be derived from equation 1:

$$s_{L_{\phi_k}}(N_{req}) = \left\lfloor \frac{L_{\phi_k}}{f_{NoC}} \times \frac{f_{mem}^c}{N_{req}} \right\rfloor \times w_{mem}^c$$

And so, the number of packets that can be read from local memory N_{pk}^{SRAM} is:

$$N_{pk}^{SRAM}(N_{req}) = \left\lfloor \frac{s_{L_{\phi_k}}(N_{req})}{s_{pk}^{max} \times s_{flit}} \right\rfloor \quad (6)$$

c) Combination: Thanks to equations 5 and 6, we can calculate the actual number of packets crossing the NoC during a time window L_{ϕ_k} with:

$$N_{pk}^{L_{\phi_k}}(N_{req}) = \min(N_{pk}^{NoC}, N_{pk}^{SRAM}(N_{req}))$$

Hence, the number of L_{ϕ_k} windows needed to send every packets of ϕ_k is:

$$N_{L_{\phi_k}}^{\phi_k}(s_{trans}, N_{req}) = \left\lceil \frac{N_{pk}^{\phi_k}(s_{trans})}{N_{pk}^{L_{\phi_k}}(N_{req})} \right\rceil$$

And the end-to-end latency t_{ϕ_k} of ϕ_k is:

$$t_{\phi_k}(s_{trans}, N_{req}) = N_{L_{\phi_k}}^{\phi_k}(s_{trans}, N_{req}) \times T_{\phi_k}$$

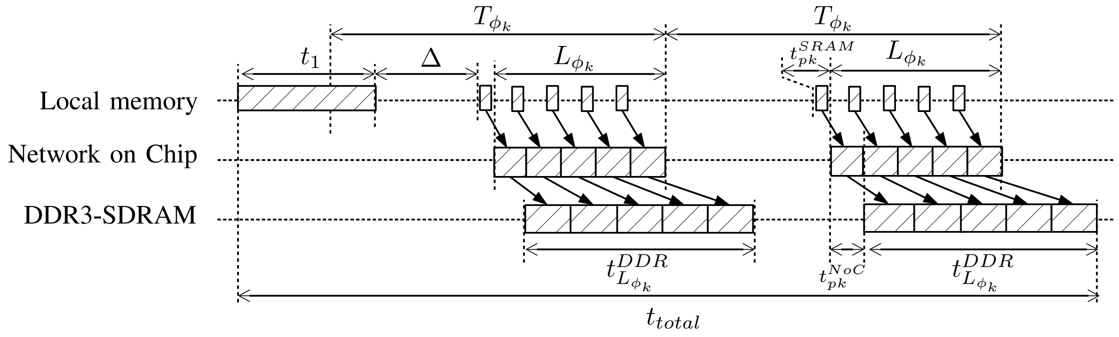


Fig. 6: End-to-end latency of a memory transaction

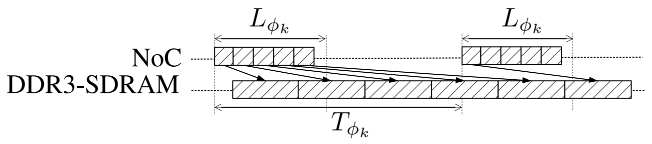


Fig. 8: Data received is not written quickly to DDR3-SDRAM

3) DDR3-SDRAM: With the bound on T_{τ}^{max} of equation 4 and the number of request per packet N_{req}^{pk}, we calculate a bound on the time required to write the $N_{pk}^{L_{\phi_k}}$ packets into memory with:

$$t_{L_{\phi_k}}^{DDR} \leq \left(\left\lceil \frac{N_{pk}^{L_{\phi_k}}}{N_{req}^{pk}} \right\rceil \times N_{trans}^{max} + 2n - 1 \right) \times t_{req}^{max} \quad (7)$$

In the rest of the paper, we denote the right part of equation 7 as $b_{L_{\phi_k}}^{DDR}$. Obviously, this bound seems pessimistic as it considers t_{req}^{max} for any request issued to the controller. However, it is a safe bound since no assumptions are made on the competitors and therefore the considered worst-case can happen. Especially, one may note that no assumptions are made on the following parameters:

- the number of competitors;
- their type (read or write);
- their locality (which row of which bank are they accessing);

4) End-to-end latency calculation: There are two cases to consider in order to calculate the maximum duration of the memory transaction:

- $b_{L_{\phi_k}}^{DDR} < T_{\phi_k}$: The data received during one L_{ϕ_k} are completely written into memory before the start of the next L_{ϕ_k}. So, the DDR3-SDRAM latency is somehow masked by the TDMA allocation of the NoC as shown in figure 6. In this case, the total end-to-end maximum duration of the memory transaction is bounded by:
$$b_{trans}^1 = t_1 + t_{\phi_k} - L_{\phi_k} + t_{pk}^{NoC} + b_{L_{\phi_k}}^{DDR}$$
- $b_{L_{\phi_k}}^{DDR} > T_{\phi_k}$: The data received from the NoC are not written into the memory fast enough and thus, are stored in a queue, waiting to be treated as shown in figure 8. We assume here that the queues are large enough to not overflow. In this case, the DDR latency is dominating the calculation of the total end to end maximum duration of the memory transaction. Its bound is:
$$b_{trans}^2 = t_1 + \Delta^{max} + t_{pk}^{SRAM} + t_{pk}^{NoC} + (N_{L_{\phi_k}}^{\phi_k} \times b_{L_{\phi_k}}^{DDR})$$

So, we can upper-bound the worst case duration of the memory transaction t_{trans}^{total} with:

$$t_{trans}^{total} \leq \max(b_{trans}^1, b_{trans}^2)$$

B. Numerical applications

We illustrate the previous analyses with an example using the indicative hardware parameters of table III. We assume a transaction of $s_{trans} = 4$ KiB of data with a corresponding flow ϕ_k crossing a path of $N_R^{\phi_k} = 4$ routers during $L_{\phi_k} = 512$ cycles every $T_{\phi_k} = 1024$ cycles.

Compute Tiles		Network on Chip		IO Tiles	
N_c^c	10	s_{flit}	4 Bytes	N_c^{io}	4
N_{dma}^c	1	s_{pk}^{max}	64 flits	N_{dma}^{io}	4
N_{bank}^c	8	s_{header}	2 flits	External Memory	
f_{mem}^c	600 MHz	f_{NoC}	600 MHz	N_{req}^{pk}	2
w_{mem}^c	8 Bytes	δ_R	5 cycles	Datasheet	[10]

TABLE III: Indicative hardware parameters

1) Local memory: We show on table IV the impact of the number of local memory competitors N_{req} on t_1 and the number of L_{ϕ_k} required to send all the data. We can see t_1 is growing linearly with N_{req}. We also note that $N_{L_{\phi_k}}$ is strongly impacted by the concurrency at local memory level. The large values of N_{req} obviously imply a large under-utilization of the NoC.

N_{req}	1	3	5	7	9	11
t_1 (in cycles)	512	1536	2560	3584	4608	5632
$N_{L_{\phi_k}}$	3	4	6	8	16	16

TABLE IV: Impact of N_{req}

2) DDR3-SDRAM: We assume t_{req}^{max} happens in the case of a row-conflicting read request following a write. Thus, using the expressions of table II, we have:

$$t_{req}^{max} = t_{WR} + t_{RP} + t_{RCD} + t_{CAS} + t_{BURST} = 67.5 \text{ ns}$$

This is required since no assumptions on the type, locality and number of concurrent transactions are made. However, the restriction of the memory access patterns thanks to an appropriate execution model could significantly decrease $b^{DDR}_{L_{\phi_k}}$. Indeed, by avoiding the overlapping of row conflicting transactions, t^{max}_{req} could be replaced by:

$$t^1_{req} = t_{CAS} + t_{BURST} = 18.75 \text{ ns}$$

and the cost of precharging and activating pages should be payed only once per transaction. This would reduce $b^{DDR}_{L_{\phi_k}}$ up to $(67.5 - 18.75)/67.5 = 72\%$. Grouping RD and WR would also allow to avoid the t_{WTR} penalty and improve both the performance and the tightness of the worst-case bound. Finally, reducing the maximum number of competitors N^{max}_{trans} will reduce N^{max}_{round} and thus $b^{DDR}_{L_{\phi_k}}$ significantly.

VI. Recommendations

A. Local memory

We have seen that the local memory of the Compute Tiles can be shared fairly amongst many requesters. However, always considering the maximum number of competitors can lead to introduce a large pessimism in the calculation of the memory accesses latencies, especially when the number of banks of the memory is close to the number of potential requesters. In this case, static bank allocation seems to be a reasonable method to bound the number of potential requesters to a bank, and thus, to reduce the implied pessimism accordingly. Moreover, we showed that the NoC utilization is strongly dependent on the local memory bandwidth allocated to the Tile's DMA. Thus, managing the maximum concurrency with the DMA seems to be a key element to ensure good performances.

B. Network on Chip

The essential three parameters for the Network on Chip management are: 1) the path allocated to each flow; 2)the width of the time window L_ϕ and 3) its corresponding period T_ϕ. The flows paths and periods must be chosen carefully. Indeed, in such strictly periodic systems, two flows with prime periods will not be able to share a NoC link [12]. So, as explained in section III-B, we recommend to use processors where the routing policy is not hardware-based but can be chosen explicitly by software to gain in flexibility. Moreover, the choice of the flows periods should not be completely unrestricted in order to avoid prime periods and to increase the number of scheduling possibilities. A reasonable approach could be to define a set of acceptable periods (that should ideally have large greatest common divisors) that any application could use. We can coarsely estimate the bandwidth allocated to one flow ϕ with $B_\phi = s_{flit} \times f_{NoC} \times \frac{L_\phi}{T_\phi}$. Hence, we can estimate a value of L_ϕ to fulfill the bandwidth requirement B_{app} of the application by $L_\phi = \frac{B_{app} \times T_\phi}{s_{flit} \times f_{NoC}}$. The exact calculation of L_ϕ will remain application dependent anyway.

C. DDR SDRAM

We have seen that mainly three parameters have an important impact on the DDR SDRAM performance and our ability to tightly bound it. Firstly, the locality of the concurrent transactions is a major parameter. We showed that private bank allocation provides good performance isolation between the competitors but obviously, when we consider a many-core processor with possibly hundreds or thousands of cores, the number of memory-requesting applications can be largely greater than the number of banks. To deal with this problem we assume each application has an allocated bank (several applications may be allocated to the same bank anyway) and we see two solutions: 1) the access to each bank is protected by a binary semaphore; 2) applications communications are activated by a pre-computed static scheduling table ensuring by construction that potential concurrent transactions do not share banks. The first solution seems to be the simplest to implement but may not provide a predictable behaviour, and so, we recommend the second one. Anyway, in both cases, in order to simplify the bank allocation, the memory controller should be configured in a non-interleaved addressing scheme so that contiguous memory addresses represent contiguous memory locations in the banks.

The maximum number of concurrent transactions is the second important parameter. We argue that decreasing the number of potential competitors can be highly beneficial. Indeed, each requester will be elected more often to place a memory request in the reordering pool, and thus, be less sensitive on the configuration of other transactions. This will improve both performance and predictability. To achieve this, the access to the external memory may be banned for some of the potential requesters (the IO Tile's cores for example). Moreover, the maximum number of requesters could be also reduced by computing a static scheduling table ensuring that the number of potential competitors is below a pre-defined trigger at any time.

Finally, the types of the concurrent transactions is the last important parameter. We explained that COTS memory controllers can largely differ in term of type management, and thus, provide fairly different performances. By considering the worst-case approach, a safe upper-bound of the memory access latency can be found. However, this bound may be large for two reasons: 1) the memory controller poorly reorders the requests and has according performances. In this case, the pessimism of the estimation is low. 2) The reordering is efficient and the estimated bound is largely superior to the actual latency. In this case, this approach introduces an important pessimism. So, the algorithm used to compute the scheduling tables should include an optimization criteria to concatenate accesses of the same type. This will both increase performance and make the memory access latency estimation insensitive to the type management policy of the controller.

VII. Related work

Many contributions in the literature propose specific memory controllers enabling predictable performance. *PREDA-*

TOR [13] uses a closed-page policy with static priority assigned to requests in order to provide bounded latency. The *Analyzable Memory Controller* (or AMC) [14] is rather similar to PREDATOR. The main difference between AMC and PREDATOR is the arbitration as AMC implements a Round-Robin arbiter. PRET [15] partitions the memory in four groups of banks (two groups by rank) and cycles through groups in a time triggered fashion in order to provide four independent resources. ROC [16] uses bank privatization to limit the impacts of row-conflicts and uses rank-switching to hide the write-to-read latencies.

However, the utilization of COTS is an important trend in industry in order to reduce both design costs and time-to market. Several contributions [17]–[20] have been proposed in the literature in order to bound the memory access latencies of multi-core processors by analyzing the DRAM access protocol and all its related timing parameters. In [17] and [18], the concurrent transactions are assumed to be reordered using a simple First-Come First-Serve (or FCFS) policy which implies a starvation-free behaviour but is not representative of real COTS memory controllers. The authors of [19] assume a First-Ready First-Come First-Serve (or FR-FCFS) policy that is largely implemented within classical COTS memory controllers but they make strong assumptions about the system model and especially the task set (each task is assumed to have enough cache space to store one row of each bank assigned to it and tasks do not share memory). In [21], the authors propose a memory bandwidth reservation system implemented as a Linux Kernel and aiming at providing guaranteed and/or best-effort memory bandwidth to the applications on COTS processors. Although the proposed approach seems to provide good performance isolation between the tasks, the bandwidth budget allocated to each core may not be respected because of a mis-prediction of the reclaim algorithm that is thus not applicable within safety critical hard real time systems.

The authors of [22] propose a global static scheduling approach to map real-time applications onto many-core processors but do not take into account the interference of potential competitors at the local memory and NoC levels. Moreover, they do not consider external resources such as the DDR-SDRAM.

VIII. Conclusion

In this paper, we proposed a realistic analysis of the sources of interference between applications on their memory access path. We defined the composition rules at the local SRAM, NoC and the DDRx-SDRAM levels in order to bound the end-to-end duration of any memory access. Hence, we quantified the potential pessimism implied by a worst-case-everywhere calculation and we proposed recommendations to choose COTS processors with specific properties and to build efficient execution models enabling a much less pessimistic estimation.

For the future, we plan to implement on real targets the proposed execution models in order to provide a formal and experimental analysis.

References

[1] Kalray, *The MPPA hardware architecture*, 2012.

[2] E. Fleury and P. Fraigniaud, “A General Theory for Deadlock Avoidance in Wormhole-Routed Networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 7, pp. 626–638, 1998.

[3] J. Duato, “A new theory of deadlock-free adaptive routing in wormhole networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 12, pp. 1320–1331, 1993.

[4] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, “Guaranteed Services of the NoC of a Manycore Processor,” in *Proceedings of the 2014 International Workshop on Network on Chip Architectures (NoCArc’14)*, 2014, pp. 11–16.

[5] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, 2001.

[6] A. Garcia, L. Johansson, M. Jonsson, and M. Weckstèn, “Guaranteed periodic real-time communication over wormhole switched networks,” in 13^{th} *Int. Conf. on Parallel and distributed computing systems (ISCA)*, 2000, pp. 632–639.

[7] JEDEC, “DDR3 SDRAM STANDARD,” 2012.

[8] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.

[9] P. Atanassov and P. Puschner, “Impact of DRAM Refresh on the Execution Time of Real-Time Tasks,” in *Proc. IEEE International Workshop on Application of Reliable Computing and Communication*, 2001, pp. 29–34.

[10] Micron, “4Gb: x4, x8, x16 DDR3L SDRAM Description,” 2011.

[11] O. Mutlu and T. Moscibroda, “Stall-time fair memory access scheduling for chip multiprocessors,” in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40. Washington, DC, USA: IEEE Computer Society, 2007, pp. 146–160.

[12] J. Korst, E. H. L. Aarts, J. K. Lenstra, and J. Wessels, “Periodic Multiprocessor Scheduling,” in *Proceedings on Parallel Architectures and Languages Europe : Volume I: Parallel Architectures and Algorithms*, ser. PARLE ’91, 1991, pp. 166–178.

[13] B. Akesson, K. Goossens, and M. Ringhofer, “PREDATOR: A Predictable SDRAM Memory Controller,” in *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS ’07, 2007, pp. 251–256.

[14] M. Paolieri, E. Quiones, F. Cazorla, and M. Valero, “An Analyzable Memory Controller for Hard Real-Time CMPs,” *Embedded Systems Letters, IEEE*, vol. 1, no. 4, pp. 86–90, 2009.

[15] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, “PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation,” in *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS ’11, 2011, pp. 99–108.

[16] Y. Krishnapillai, Z. P. Wu, and R. Pellizzoni, “A Rank-Switching, Open-Row DRAM Controller for Time-Predictable Systems,” in *26th Euromicro Conference on Real-Time Systems (ECRTS’14)*, 2014, pp. 27–38.

[17] Y. Ding, L. Wu, and W. Zhang, “Bounding Worst-Case DRAM Performance on Multicore Processors,” *Jour. of Comp. Science and Engineering*, vol. 7, no. 1, pp. 53–66, 2013.

[18] Z. P. Wu, Y. Krish, and R. Pellizzoni, “Worst Case Analysis of DRAM Latency in Multi-requestor Systems,” in *34th Real-Time Systems Symposium (RTSS’13)*, 2013, pp. 372–383.

[19] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. R. Rajkumar, “Bounding memory interference delay in COTS-based multi-core systems,” in *20th Real-Time and Embedded Technology and Applications Symposium (RTAS’14)*, 2014.

[20] H. Yun and R. Pellizzoni, “Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems,” 2014, submited (arXiv).

[21] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, “MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms,” in *19th Real-Time and Embedded Technology and Applications Symposium (RTAS’13)*, 2013, pp. 55–64.

[22] T. Carle, M. Djemal, D. Potop-Butucaru, and R. De Simone, “Static mapping of real-time applications onto massively parallel processor arrays,” in *14th International Conference on Application of Concurrency to System Design*, ser. Proceedings ACSD 2014, Hammamet, Tunisia, 2014.

Session 15

Test

Thursday 28th, 11:45 – Ariane 1

Automatic Interleaving for Testing Distributed Systems

Mihal Brumbulli and Emmanuel Gaudin
PragmaDev
{mihal.brumbulli,emmanuel.gaudin}@pragmadev.com

***Abstract*—The constantly ever-growing interest for large-scale distributed systems like the Internet of Things imposes many challenges for developers and researchers from many areas. The development of distributed software applications is by no means trivial, and their inherent complexity becomes apparent during testing. Indeed, testing the operation of single isolated nodes does not suffice, because it may be affected by the distribution and inter-communication between nodes. Re-writing a test case to consider distribution is neither efficient nor simple, because concurrency is never easy to implement. In this paper we present an approach that automatically interleaves execution of test cases to simulate concurrency inherent in distribution. We focus on independent test cases that might exhibit a correlation due to distributed interaction. The approach is applied in the context of standard modeling and testing languages, and enables identification of interaction points during test case execution that depend on distribution. The re-execution of the test case is then interleaved at the identified points to account for distribution.**

***Index Terms*—Distributed systems, testing, modeling, simulation, TTCN-3, SDL**

I. Introduction

The development of distributed systems has gained much attention from industry and researchers with a variety of applications, a trend that is sure to continue in the immediate future due to the ever-growing interest for the Internet of Things [1]. However, the development of software applications for large-scale distributed systems is not a trivial task. This is especially true for testing, an activity which is quite challenging even for simple non-distributed systems. The operation of nodes in a distributed system is not isolated, and as such it requires for test cases to account for the distribution and interaction between nodes. The implication here is that existing test cases and their execution have to be adapted to consider distribution. This adaptation consists of (a) introducing concurrency handling into test cases, and (b) controlled concurrent execution that deals with all relevant interleavings. For the former the tests cases have to be modified, and considering that concurrency handling is never easy to implement, the effort that is required should not be overlooked. The later implies the existence of a scheduler that is able to handle all relevant interleavings.

In this paper we present an approach for automating the interleaved execution of test cases. We apply the approach in the context of standard modeling and testing languages. The system under test is described in SDL [2], TTCN-3 [3] is used for testing, and SDL-RT deployment diagrams [4] describe the distribution of system components. Test cases are executed against the system in a simulated environment extended with an interleaving algorithm.

In Section II we give an overview of related work and position our approach in respect to existing state of the art. We introduce the relevant technologies in Section III and our approach in Section IV. An example is given in Section V to illustrate the use of the presented solution. Finally, we conclude in Section VI with a discussion around the approach, its current status, and future work.

II. Related Work

Testing of distributed communication systems has been approached from different angles. We identify two major groups, i.e., distributed testing and interleaved execution.

Hartman et al. in [5] discuss the execution of abstract tests for distributed software. They underline the importance and benefits of interleaving test case sequences for discovering defects in the system. The approach can be applied to a single test case, it uses either concurrent or sequential synchronization of execution sequences, and there is no automation involved in establishing the synchronization points.

Schieferdecker and Vassiliou-Gioles in [6] discuss the distribution of TTCN-3 test setups. The focus is on the underlying concepts of the language and how they support management and execution of distributed test cases. This approach implies test execution on the target. This is obviously an advantage, however, it also needs a complex synchronization mechanism for controlling execution of each of the distributed test cases.

Bloom et al. in [7] emphasize the fact that, prior to testing in a target environment, the software is usually tested in the host environment. They propose a simulation-based approach and focus on the semantics of time. However, it is not clear how the execution of several test instances is interleaved to account for the inherent concurrency.

Testing of concurrent software is discussed in [8] and [9]. The authors focus on multi-threaded software and propose a solution that interleaves execution in a controlled way. The approach uses neither a standard language (like TTCN-3) nor any kind of abstract notations, instead, it is based on general purpose programming languages.

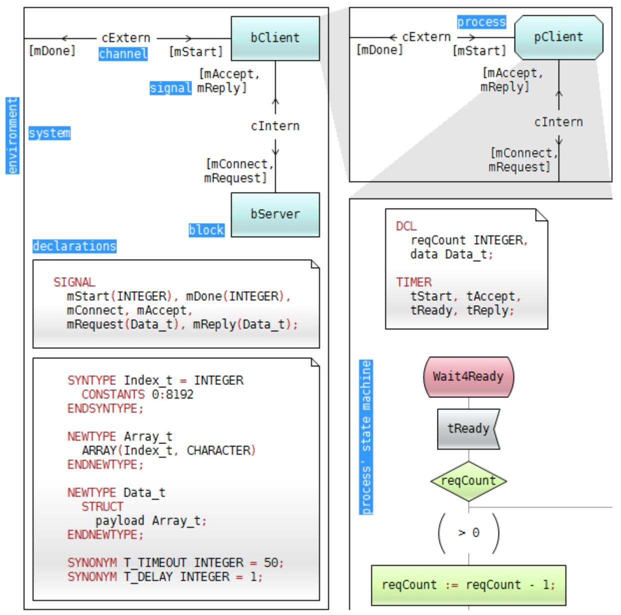
We adopt the idea of controlled interleaved execution in the context of distributed systems. To do so, we consider distributed communication between instances of the system to be similar to thread interleavings, and can potentially impact the behavior induced by the execution of test cases. Furthermore, the approach is based on standard and formal languages with precise semantics. This allows simulation in a controlled environment and, what is of great interest, automatic generation and execution of all relevant interleavings.

III. Technology

A. SDL

The Specification and Description Language (SDL) is a specification language defined by the International Telecommunication Union (ITU-T) in the Z.100 series [2]. SDL is targeted at the unambiguous specification and description of the behavior of reactive and distributed systems.

1) Structure, Behavior, and Communication: In SDL the overall design is called the *system*, and everything outside of it is defined as the *environment*. The system can be composed of *agents* and *communication constructs*. There are two kinds of agents: *blocks* and *processes*. Blocks can be composed of other agents and communication constructs. When the system is decomposed down to the simplest block, the way the block fulfills its functionality is described with processes. A process provides this functionality via extended finite state machines. It has an implicit queue for *signals*. A signal has a name and parameters; they go through *channels* that connect agents and end up in the processes implicit queues. Fig. 1 illustrates these concepts in a simple client-server application example.



```
SIGNAL
  mStart(INTEGER), mDone(INTEGER),
  mConnect, mAccept,
  mRequest(Data_t), mReply(Data_t);
```

```
SYNTYPE Index_t = INTEGER
  CONSTANTS 0:8192
ENDSYNTYPE;

NEWTYPE Array_t
  ARRAY(Index_t, CHARACTER)
ENDNEWTYPE;

NEWTYPE Data_t
  STRUCT
    payload Array_t;
ENDNEWTYPE;

SYNONYM T_TIMEOUT INTEGER = 50;
SYNONYM T_DELAY INTEGER = 1;
```

```
DCL
  reqCount INTEGER,
  data Data_t;

TIMER
  tStart, tAccept,
  tReady, tReply;
```

Fig. 1. Excerpt of the SDL model of a client-server application.

The client is started with *mStart* signal which takes as parameter the number of request the client should send to the server. The client will try to connect to the server, and in case of success it will start sending requests and waiting for replies. When done (or in case of error) it will report back (to the environment) the number of replies received from the server via the *mDone* signal.

2) Deployment: SDL descriptions are platform independent, i.e., they do not capture any information concerning the implementation details. For example, Fig. 1 speaks of a *bClient* and *bServer*, however it does not specify whether these agents are distributed or not. Deployment diagrams as defined in [4] can supplement the models with missing information about distribution. This approach has been used for simulating distributed applications as described in [10] and [11]. We adopt the idea and simplify it by focusing only on *components*, *nodes*, and *connections* as shown in Fig. 2.[1] The semantics are straightforward, i.e., if components (representing SDL agents) are attached to different nodes, then they are considered distributed, otherwise local execution is implied.

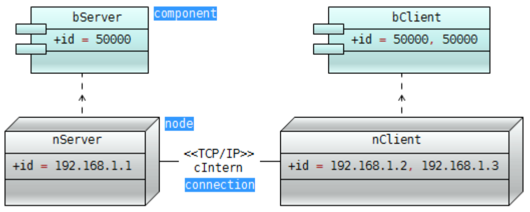


Fig. 2. Deployment model of the client-server application.

B. TTCN-3

The Testing and Test Control Notation Version 3 (TTCN-3) is a standardized testing technology developed and maintained by the European Telecommunication Standards Institute (ETSI). The ETSI TTCN-3 standards [3] have also been adopted by the ITU-T in the Z.160 series [12].

The abstract definition of test cases makes it possible to specify a non-proprietary test systems which are independent of both platform and operating system. The abstract definitions can be either compiled or interpreted for execution.

Fig. 3 shows a TTCN-3 module definition with a single test case that triggers the sending of 10 requests (line 15 and 19) from the client and expects 10 replies (line 16 and 21).

```
 1: module TestClientServer {
 2:   // Types for messages
 3:   type record mStart { integer reqCount };
 4:   type record mDone { integer repCount };
 5:   // Port type for the interface with the SUT
 6:   type port port_cExtern message {
 7:     out mStart;
 8:     in mDone;
 9:   };
10:   // Component type for the MTC and system inteface
11:   type component sClientServer {
12:     port port_cExtern cExtern;
13:   };
14:   // Templates for messages
15:   template mStart startMessage := { reqCount := 10 };
16:   template mDone doneMessage := { repCount := 10 };
17:   // Testcase
18:   testcase tc_start_done() runs on sClientServer {
19:     cExtern.send(startMessage);
20:     alt {
21:       [] cExtern.receive(doneMessage) {
22:         setverdict(pass);
23:       }
24:       [] cExtern.receive {
25:         setverdict(fail);
26:       }
27:     }
28:   }
29: }
```

Fig. 3. TTCN-3 module with a test case for the client-server application.

[1]The number of values in the *id* attribute is the number distributed nodes, e.g., the figure implies two clients and one server.

IV. Approach

A. Problem Statement

We are interested in the effects (if any) induced by distributed execution of test cases. To better understand the problem let us revisit the client-server example, and suppose that the result of the test case presented in Fig. 3 is *pass*, i.e., the system behaves as expected. What will happen if the number of clients is increased; will the system behave the same? There is one effective way to answer this question: test the system with multiple clients. This can be tackled by (a) rewriting the test case so that it accounts for multiple clients, or (b) execute multiple instances of the test case (one for each client) in parallel. The later needs an underlying synchronization mechanism that allows controlled execution of parallel (distributed) test cases, which is never trivial and requires additional expertise (not TTCN-3). That is why the former is more sound from a tester's perspective, because it is confined at the abstract level of testing provided by TTCN-3. However, rewriting the test case to take into account only the number of clients is not enough. Indeed, the difficult part is not in the number of clients but in their parallel (concurrent) execution due to distribution. To model concurrency all possible *interleavings* between test cases should be considered, which implies execution of all permutations of TTCN-3 instructions of different clients. For example, if *startMessage* is sent first to *client_1* and then to *client_2*, it is important to consider also the case where it is first sent to *client_2* and then to *client_1*.

B. Problem Analysis

Supposing concurrent execution of K test cases, with each test case consisting of n_i instructions for $i = 1, 2, \ldots, K$, the number of all interleavings is given by:

$$I = \frac{(\sum_{i=1}^{K} n_i)!}{\prod_{i=1}^{K} (n_i!)} \tag{1}$$

If we consider concurrent execution of K instances of the same test case, then $n_i = N \ \forall i$, where N is the number of instructions in the test case, and (1) can be re-written as:

$$I = \frac{(KN)!}{(N!)^K} \tag{2}$$

This is a typical case of the *state-explosion problem* which makes execution of all interleavings unpractical even in a controlled and automated environment. However, not all interleavings are relevant and their number (in most cases) can be drastically reduced. Indeed, the behavior induced by a test case can be affected by distribution only if there is an interaction between nodes. This means that, if the execution of a test case does not involve any distributed communication, then distribution will not have any impact. For example, if there isn't any communication between the client and the server, then interleaving is pointless because there is no distributed behavior in the first place. That is why it makes sense to interleave execution at critical points, i.e., instructions that trigger interaction between nodes by means of distributed communication.

C. Interleaving Algorithm

We start by grouping the instructions and then interleaving the execution of the groups. The condition is that each group must include at most one instruction which triggers distributed communication. Let m_i^j be an instruction in the test case, where $i = 1, 2, \ldots, N$ is the index (relative order) of the instruction, and $j = 0, 1$ describes whether the given instruction triggers any interaction (interleaving point). A group consists of all subsequent m_i^j for which $\sum j \leq 1$. The following shows an example sequence and corresponding grouping:

$$\underbrace{m_1^0, m_2^1, m_3^0}_{g_1}, \underbrace{m_4^1}_{g_2}, \underbrace{m_5^1, m_6^0}_{g_3}, \underbrace{m_7^1, m_8^0, m_9^0}_{g_4}, \underbrace{m_{10}^1}_{g_5} \tag{3}$$

For two test cases with the above sequence, the algorithm can generate > 700 times less interleavings.[2]

Interleaved execution can be automated using the following algorithm:

```
{$K$ is the number of instances}
{$N$ is the number of groups}
{$I$ is the next interleaving}
**for** $i := 0$ **to** $K - 1$ **do**
  **for** $j := 0$ **to** $N - 1$ **do**
    $I[i * N + j] := i$
  **end for**
**end for**
**loop**
  **handleInterleave**($I$)
  $i := I.length - 1$
  **while** $i > 0$ **and** $I[i-1] \geq I[i]$ **do**
    $i := i - 1$
  **end while**
  **if** $i = 0$ **then**
    **return**
  **end if**
  $j := I.length - 1$
  **while** $I[j] \leq I[i-1]$ **do**
    $j := j - 1$
  **end while**
  $temp := I[i-1]$
  $I[i-1] := I[j]$
  $I[j] := temp$
  $j := I.length - 1$
  **while** $i < j$ **do**
    $temp := I[i]$
    $I[i] := I[j]$
    $I[j] := temp$
    $i := i + 1$
    $j := j - 1$
  **end while**
**end loop**
```

[2]The result can be obtained by replacing in (2): $K = 2$ and $N = 10$ prior to grouping, and $K = 2$ and $N = 5$ after grouping.

Each group of instructions is represented by a simple integer for ease of calculation. The *handleInterleave* procedure maps the integer to its corresponding sequence of TTCN-3 instructions (permutation) that is to be executed next.

D. *Tool Support*

PragmaDev Studio[3] is a set of tools that helps specifiers, developers, and testers to manage complexity in the development of today's systems. The tools use the recognized international standards of SDL, TTCN-3, SDL-RT, and UML [13].

A key functionality of the tool-set is provided by the *PragmaDev (Co-) Simulator* as shown in Fig. 4.

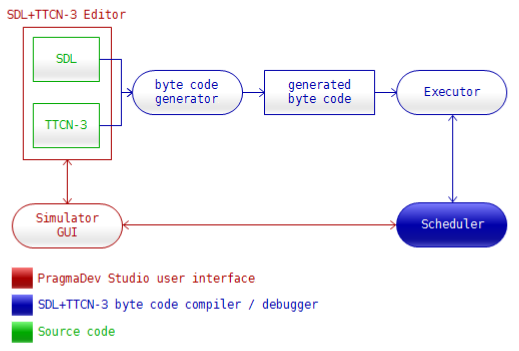


Fig. 4. Architecture of the PragmaDev (Co-) Simulator.

The co-simulator allows execution of TTCN-3 test cases against an SDL system. SDL and TTCN-3 descriptions are translated into an internal representation (byte code) to be interpreted by the *executor*, which in turn forwards the scheduling of events to the *scheduler*. We extend the functionality of the scheduler with the interleaving algorithm.

In the first phase the test case is executed against the system using the scheduler in normal mode (no interleaving involved). All TTCN-3 instructions (*send* statements) that trigger distributed communication between agents of the SDL system are marked during execution. Every communication between agents is checked against a deployment diagram, and if the sender and receiver of a message are attached to different nodes in the diagram, then the last TTCN-3 instruction that triggered such behavior is indeed the one to be marked.

In the second phase the test case is executed against the system in interleaving mode. The scheduler automatically creates K instances[4] of the test case and enters the interleaving algorithm. On each iteration the algorithm creates an interleaved sequence of TTCN-3 instructions based on marking done in the first phase and the number of instances. The sequence is then executed like a normal TTCN-3 test case by the scheduler, and at the end the SDL system is reset to its initial state for the next iteration.

The whole process is completely automated and transparent to the tester. There is no need to rewrite the tests, but just execute them with the interleaving scheduler.

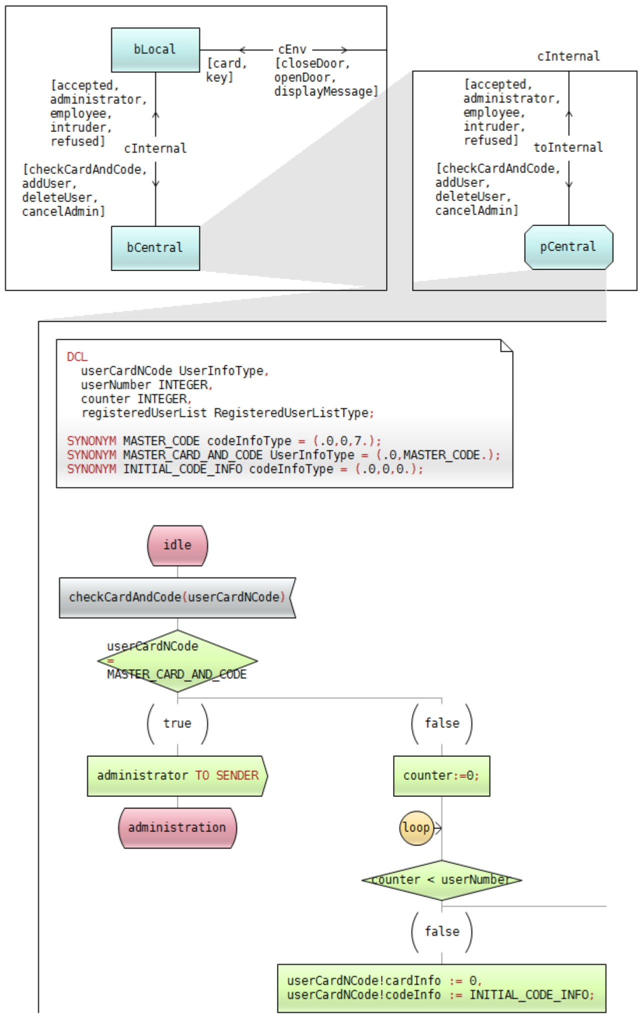
[3]http://www.pragmadev.com
[4]The number of instances is deduced from the deployment diagram.

V. Example

We have used the presented approach in testing an access-control system. The system is composed of several terminals and a central unit. Each terminal has a slot for entering a card and a keypad for entering the key. This information is sent to the central unit which checks whether access should be granted to a user and notifies the terminal from where the request was issued. The user can be either an *administrator* or a *normal* one. The administrator can add or delete normal users and is identified by a special card key.

A. *Structure and Behavior*

Fig. 5 shows an excerpt of the SDL model of the access-control system. The system is composed of two blocks, namely *bLocal* for terminals and *bCentral* for the central unit. Each block has a single process within which implements the behavior of the system.


```
DCL
  userCardNCode UserInfoType,
  userNumber INTEGER,
  counter INTEGER,
  registeredUserList RegisteredUserListType;

SYNONYM MASTER_CODE codeInfoType = (.0,0,7.);
SYNONYM MASTER_CARD_AND_CODE UserInfoType = (.0,MASTER_CODE.);
SYNONYM INITIAL_CODE_INFO codeInfoType = (.0,0,0.);
```
```
userCardNCode!cardInfo := 0,
userCardNCode!codeInfo := INITIAL_CODE_INFO;
```
bLocal
[card, key]
cEnv
[closeDoor, openDoor, displayMessage]
[accepted, administrator, employee, intruder, refused]
cInternal
[checkCardAndCode, addUser, deleteUser, cancelAdmin]
bCentral
cInternal
[accepted, administrator, employee, intruder, refused]
toInternal
[checkCardAndCode, addUser, deleteUser, cancelAdmin]
pCentral
idle
checkCardAndCode(userCardNCode)
userCardNCode = MASTER_CARD_AND_CODE
true
false
administrator TO SENDER
counter:=0;
administration
loop
counter < userNumber
false

Fig. 5. Excerpt of the SDL model of the access-control system.

The user enters the *card* and the *key* which are sent to the central unit via the *checkCardAndCode*. If the card and key are those of the administrator, then the central unit enters in *administration* state, in which normal users can be added or deleted. On the other hand, if the credentials are not those of the administrator, the list of registered normal users is scanned for matching credentials. The *employee* signal is sent back in case of a match, otherwise an *intruder* is signaled.

B. Deployment

A simple deployment scenario for the access-control system is shown in Fig. 6. The figure speaks of two *bLocal* (terminals) connected to a single *bCentral* (central unit). We will use this scenario to automatically generate and execute all relevant interleavings.

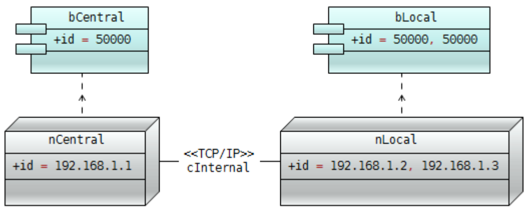


Fig. 6. Deployment model of the access-control system.

C. Test Case

To show the applicability of automatic interleaving we start with the most basic test case for the system, i.e., try to get in and out of (without doing anything else) administrator mode as shown in Fig. 7.

```
 1: module TestAccessControl {
 2:   type record card { integer param1 };
 3:   type record key { integer param1 };
 4:   type record displayMessage { charstring param1 };
 5:   type record openDoor { charstring param1 };
 6:   type record closeDoor { charstring param1 };
 7:   type port cEnv_type message {
 8:     out card;
 9:     out key;
10:     in displayMessage;
11:     in closeDoor;
12:     in openDoor;
13:   };
14:   type component AccessControl {
15:     port cEnv_type cEnv;
16:   };
17:
18:   template displayMessage EnterCardMessage := {param1 := "Enter card"};
19:   template displayMessage EnterCodeMessage := {param1 := "Enter code"};
20:   template displayMessage AddOrDeleteMessage := {param1 := "* add; # delete"};
21:   template displayMessage OneStar := {param1 := "*"};
22:   template displayMessage TwoStar := {param1 := "**"};
23:   template displayMessage CancelledMessage := {param1 := "Cancelled"};
24:   template card UserCard(integer userID) := { param1 := userID };
25:   template key EnterKey(integer keyValue) := { param1 := keyValue };
26:
27:   altstep failOnWrongReceive() runs on AccessControl {
28:     [] cEnv.receive { setverdict(fail); };
29:   }
30:
31:   testcase tc_correctAdministratorAccess() runs on AccessControl {
32:     activate(failOnWrongReceive());
33:     // Enter card
34:     cEnv.receive(EnterCardMessage);
35:     cEnv.send(EnterCard(0));
36:     // Enter administrator code
37:     cEnv.receive(EnterCodeMessage);
38:     cEnv.send(EnterKey(0));
39:     cEnv.receive(OneStar);
40:     cEnv.send(EnterKey(0));
41:     cEnv.receive(TwoStar);
42:     cEnv.send(EnterKey(7));
43:     // Administrator mode
44:     cEnv.receive(AddOrDeleteMessage);
45:     // Get out of administrator mode
46:     cEnv.send(EnterKey(0));
47:     cEnv.receive(CancelledMessage);
48:     // Done
49:     setverdict(pass);
50:   }
51: }
```

Fig. 7. Test case for administrator access on the access-control system.

The user enters the *card* = 0 (administrator) and then the *code* = 007. At this point the information (card + code) is sent to the central unit. Because these are the right credentials, the user is allowed to enter in administrator mode, where he/she can add or delete users. However, none of this actions is taken, and the user just exits from the administrator mode. The request to exit is also processed by the central unit. Executing this scenario against the system using the scheduler in normal mode (no interleaving, one terminal and one central unit) will indeed result in a *pass* for the test case.

When execution in normal mode is finished all the required information will be available for entering the interleaving mode. Based on the description above there should have been identified two interleaving points by now: (1) after entering the last digit of the code and (2) after the request to leave the administrator mode. Indeed, these are the points during execution where a communication between the terminal *bLocal* and the central unit *bCentral* is triggered. This translates into two groups of TTCN-3 statements whose execution shall be interleaved: the first group consists of lines 34-44 and the second of lines 46-47 in Fig. 7. Based on the number of terminals *bLocal* in Fig. 6 and that of groups ($K = 2$, $N = 2$), the total number of interleavings to be executed is 6. It is important to note that, if the grouping algorithm is not applied, the number of interleavings will be 252.[5]

What we found during execution in interleaving mode was in fact quite surprising. We didn't expect to find any problems from such a simple test case, considering that it induces a very basic behavior to the system. The system was modeled so that only one terminal at a time can get administrator access, meaning that an attempt from a second terminal will fail. Indeed, this is the behavior we observed during the interleaved execution, however, what we did not expect is for the second terminal to block indefinitely waiting for a reply from the central unit (i.e., test case execution did block on line 44 in Fig. 6). We were able to immediately jump to the point in the model (using the PragmaDev Studio user interface) causing the problem. The central unit, after granting access to the first terminal, entered the *administration* state so that it could process any requests coming from the terminal for adding or deleting users. However, in this state it was discarding (not handling) all log-in requests, which in turn caused the second terminal to wait indefinitely for a reply. The solution was straightforward: add a new transition in the SDL state machine of the central unit to handle such requests.

We were able to identify 4 similar problems in our model of the access-control system by using the interleaving scheduler on a set of more complex test cases. The set was composed of (a) previously hand written test cases depicting typical usage scenarios and (b) a set of automatically generated test cases using the model-based approach described in [14]. It is important to note that we were able to identify these problems without writing a single line of TTCN-3 code.

[5]In this case the number of groups is equal to that of the TTCN-3 *send* statements ($N = 5$).

VI. Conclusion

Testing of software applications for large-scale distributed systems is not a trivial task, because the test cases and their execution need to be adapted in order to account for the distribution and interaction between nodes.

In this paper we presented an approach for automating the interleaved execution of test cases to mitigate the complexity of concurrent behavior due to distribution. We applied the approach in the context of standard modeling and testing languages where the system under test was described in SDL, the distribution of system components in deployment diagrams, and the test cases in TTCN-3. Test cases were executed against the system in a simulated and controlled environment with an interleaving scheduler.

The presented algorithm allowed us to significantly reduce the number of interleavings without deviating from our target, that is the thorough testing of the system. Instead of using each single instruction of the test case as interleaving point, we focused only on those relevant, i.e., instructions that induce distributed behavior to the system.

Automatic interleaving was used for testing the distributed behavior of an access-control system. Even with a basic test case we were able to identify a problem in the system, a trend that continued also with more complex test cases. Furthermore, we achieved these results by reducing the number of interleavings by a factor of 42 (in the simplest case).

There are however three issues that need further discussion. First, the proposed algorithm may not always produce significantly less interleavings. This depends on the degree of distribution in the behavior, i.e., the number of interleavings grows if more inter-communication between distributed nodes takes place. This can degrade simulation performance especially with the increasing complexity of test cases. However, we believe this type of scenario to be more of an exception than the rule. Indeed, energy consumption is one of the major challenges of distributed systems composed of potentially millions of battery powered devices (e.g., Internet of Things), and reducing inter-communication between devices to an acceptable minimum is always an engineering goal. Second, although we reported a successful application of the approach with a simple example, we strongly believe that its benefits are emphasized when used for testing complex systems, which is still work in progress as the time of writing this paper. Last, the proposed approach is based on simulation and at present cannot be applied for test cases executing on target. The reason is simple: simulation allows fine-grained control over the execution, does not require any complex synchronization mechanism for interleaving, and can exploit the benefits of the proposed algorithm. The last point is very important because, without any means of reducing the number of interleavings, testing every possible combination is unpractical due to the state-explosion problem. In the end, in addition to a mechanism for controlling the interleaving, execution on target would require means to track distributed communication triggered during testing.

References

[1] Gartner Inc., "Gartner says the Internet of Things installed base will grow to 26 billion units by 2020," http://www.gartner.com/newsroom/id/2636073, 2013.

[2] ITU-T, "Specification and Description Language – Overview of SDL-2010," International Telecommunication Union – Telecommunication Standardization Sector, ITU-T Recommendation Z.100, 2011, http://handle.itu.int/11.1002/1000/11387.

[3] ETSI, "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language," European Telecommunications Standards Institute, ETSI Standard ES 201 873-1, 2014, http://www.ttcn-3.org/index.php/downloads/standards.

[4] SDL-RT Consortium, "Specification and Description Language – Real Time," SDL-RT Consortium, SDL-RT Standard V2.3, 2013, http://www.sdl-rt.org/standard/V2.3/html/index.htm.

[5] A. Hartman, A. Kirshin, and K. Nagin, "A Test Execution Environment Running Abstract Tests for Distributed Software," in *Proceedings of Software Engineering and Applications*, ser. SEA '02. Acta Press, 2002.

[6] I. Schieferdecker and T. Vassiliou-Gioles, "Realizing Distributed TTCN-3 Test Systems with TCI," in *Testing of Communicating Systems*, ser. Lecture Notes in Computer Science, D. Hogrefe and A. Wiles, Eds. Springer Berlin Heidelberg, 2003, vol. 2644, pp. 95–109.

[7] S. Blom, T. Deiß, N. Ioustinova, A. Kontio, J. van de Pol, A. Rennoch, and N. Sidorova, "TTCN-3 for Distributed Testing Embedded Software," in *Perspectives of Systems Informatics*, ser. Lecture Notes in Computer Science, I. Virbitskaite and A. Voronkov, Eds. Springer Berlin Heidelberg, 2007, vol. 4378, pp. 98–111.

[8] M. Musuvathi and S. Qadeer, "CHESS: Systematic Stress Testing of Concurrent Software," in *Logic-Based Program Synthesis and Transformation*, ser. Lecture Notes in Computer Science, G. Puebla, Ed. Springer Berlin Heidelberg, 2007, vol. 4407, pp. 15–16.

[9] ——, "Iterative Context Bounding for Systematic Testing of Multithreaded Programs," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, J. Ferrante and K. S. McKinley, Eds. ACM, 2007, pp. 446–455.

[10] M. Brumbulli and J. Fischer, "Simulation Configuration Modeling of Distributed Communication Systems," in *System Analysis and Modeling: Theory and Practice*, ser. Lecture Notes in Computer Science, Ø. Haugen, R. Reed, and R. Gotzhein, Eds. Springer Berlin Heidelberg, 2013, vol. 7744, pp. 198–211.

[11] M. Brumbulli, "Model-Driven Development and Simulation of Distributed Communication Systems," Ph.D. dissertation, Humboldt Universität zu Berlin, 2015.

[12] ITU-T, "Testing and Test Control Notation version 3: TTCN-3 core language," International Telecommunication Union – Telecommunication Standardization Sector, ITU-T Recommendation Z.160, 2014, http://handle.itu.int/11.1002/1000/12346.

[13] OMG, "OMG Unified Modeling Language (OMG UML). Version 2.5," Object Management Group, OMG Standard, 2015.

[14] J. Deltour, A. Faivre, E. Gaudin, and A. Lapitre, "Model-Based Testing: An Approach with SDL/RTDS and DIVERSITY," in *System Analysis and Modeling: Models and Reusability*, ser. Lecture Notes in Computer Science, D. Amyot, P. Fonseca i Casas, and G. Mussbacher, Eds. Springer International Publishing, 2014, vol. 8769, pp. 198–206.

Facing ADAS validation complexity with usage oriented testing

Authors: Laurent Raffaëlli and Frédérique Vallée, All4tec ; Guy Fayolle, Inria-Armines ; Philippe De Souza, ESI ; Xavier Rouah, Intempora ; Matthieu Pfeiffer, Magillem ; Stéphane Géronimi, PSA ; Frédéric Pétrot, TIMA ; Samia Ahiad, Valeo

Keywords: Model-Based-Testing (MBT), Test Optimization, Advanced Driver Assistance Systems (ADAS), safety, ISO 26262

1 Scope

Validating Advanced Driver Assistance Systems (ADAS) is a strategic issue, since such systems are becoming increasingly widespread in the automotive field. ADAS bring extra comfort to drivers, and this has become a selling point. But these functions, while useful, must not affect the general safety of the vehicle which is the manufacturer's responsibility.

A significant number of current ADAS are based on vision systems, and applications such as obstacle detection and detection of pedestrians have become essential components of functions such as automatic emergency braking. These systems that preserve and protect road users take on even more importance with the arrival of the new Euro NCAP protocols.

Therefore the robustness and reliability of ADAS functions cannot be neglected and car manufacturers need to have tools to ensure that the ADAS functions running on their vehicles operate with the utmost safety.

Furthermore, the complexity of these systems in conjunction with the nearly infinite number of parameter combinations related to the usage profile of functions based on image sensors push us to think about testing optimization methods and tool standards to support the design and validation phases of ADAS systems. The resources required for the validation using current methods make them actually less and less adapted to new active safety features, which induce very strong dependability requirements.

Today, to test the camera-based ADAS, test vehicles are equipped with these systems and are performing long hours of driving that can last for years. These tests are used to validate the use of the function and to verify its response to the requirements described in the specifications without considering the functional safety standard ISO26262.

Therefore there is also a need to improve the way of validating the ADAS functions.

2 The COVADEC project

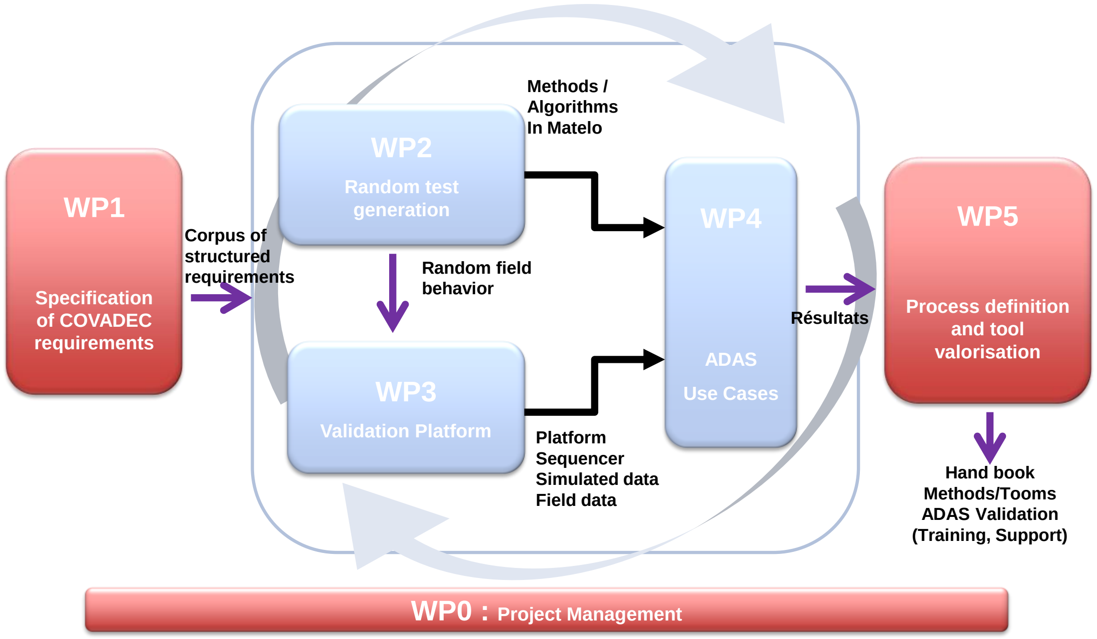
The French research & development project COVADEC(*), started in the mid-2013 aims to provide methods and techniques for automotive OEMs and suppliers who face these problems.

(*) COVADEC stands for « Conception et Validation des Systèmes Embarqués d'Aide à la Conduite » which means « Design and Validation of ADAS » in French.

COVADEC main objectives are:

- Optimize test scenarios and reduce the hundreds of thousands of kilometres of driving required for the validation of ADAS functions integrated in vehicles.
- Optimize time consumption and human effort during the validation phases of ADAS.
- Meet the needs of ADAS in terms of compliance with standards such as ISO26262 or compliance with targets of occurrence of dangerous events.
- Take into account the dependability requirements upstream of the development of image processing algorithms.
- Standardize methods and tools required for the validation of functional requirements and operational safety.
- Enhance the development of ADAS functions by anticipating and implementing in priority critical situations that can default driver assistance systems.
- Ensure interoperability between test platforms, simulation platforms (PRO-SIVIC) and other development platforms (RTMaps, ADTF).

The ADAS domain is currently not properly covered by the ISO 26262 safety standard requirements, so one of the goals of the project is to propose new solutions that can fit into the validation process established for any automotive system in order to complement the ISO 26262 gaps on the subject.


WP1
Specification of COVADEC requirements
Corpus of structured requirements
WP2
Random test generation
Methods / Algorithms In Matelo
Random field behavior
WP3
Validation Platform
Platform Sequencer Simulated data Field data
WP4
ADAS Use Cases
Résultats
WP5
Process definition and tool valorisation
Hand book Methods/Tooms ADAS Validation (Training, Support)
WP0 : Project Management

COVADEC is decomposed into 5 work packages:

- WP0: Project Management
- WP1: Specification of COVADEC requirements

The main objectives of WP1 are:

(i) collect and synthesize the needs of stakeholders in the automotive industry in terms of certification and validation, especially for driving assistance systems based on the perception of the vehicle environment

(ii) identify common practices of validation management and express the expectations of the industry for process improvement

(iii) specify the architecture of the solutions proposed in response to these expectations

- WP2: Random test generation :
  WP2 aims at developing a methodology and the associated tools to generate statistical tests meeting the requirements of COVADEC project.

- WP3: Validation Platform :
  WP3 will develop and implement software tools for the execution of test cases and the generation of test reports as generated by WP2 tools.

- WP4: ADAS Use Cases :
  Two use cases have been chosen in order to demonstrate the validity of COVADEC method and platform:
  - A Lane Departure Warning (LDW) function which informs the driver from unwanted lane departure.
  - An Automatic Emergency Braking (AEB) function which prevents from front collisions.

- WP5: Process definition and tool valorisation :
  WP5 objectives are to extend the results of the case studies examined in COVADEC to all ADAS applications; to make available to the ADAS community the COVADEC results and to explain how these results will be reused and valued by COVADEC partners.

3 Positioning with regards to the state of the art and to ISO 26262

The stakes of functional safety, in order to guarantee safety of people and goods, is to evaluate intrinsic risks of the system and thus provide solutions to reduce the probability of hazards occurrence. The subject has been extensively covered and different standards have emerged including the ISO 26262 standard in 2011 in the automotive field, standard that can be applied to ADAS programmable electronics for automotive systems.

Regarding ADAS, the main safety risks concern an erroneous analysis of a driving situation, which may provide incorrect information to the driver or, even worse, trigger an automatic and inappropriate response of the vehicle. For instance, for the Lane Departure functions, if the ADAS is coupled to the steering control of the vehicle (Lane Keeping Assist), the system may cause the vehicle to steer unwillingly or, in the case of the Automatic Emergency Braking System, brake unwillingly.

Actually the current version of the norm ISO 26262-2011 explicitly excludes dangers that are inherent to the nominal behaviour of systems i.e. when potential hazards are caused by intrinsic limitations of sensors performances (because of their probabilistic responses), and are not a result of system failures.

After further analysis, the erroneous decisions taken by the ADAS are considered as nominal behaviour of these systems and consequently not covered by this norm. Hence, the most critical cases are not Hardware or Software failures, which are already covered by existing norms i.e. ISO 26262, but the cases in which the behaviour is diverging from the nominal behaviour. In this case, the treatment and analysis chain doesn't contain any malfunction, but produces an erroneous decision.

Such a wrongful decision implies two kinds of feared events:

- the system does not detect a dangerous situation that should have been detected as such.
- the system detects a dangerous situation when this is not the case in reality

For Automatic Emergency Braking Function, the first case is less critical in terms of functional safety, as the driver may always take the good decision and control the vehicle, while the ADAS does not trigger any action. In this case, the quality of service of the function is diminished, but there is no additional danger involved by the use of the system. On the other hand, the second case is critical, as the ADAS generates wrong information or triggers a wrong action for the vehicle. In this case, using the vehicle with the ADAS may be more dangerous than without it.

In order to carry out the validation of ADAS and their robustness against feared events, different methodologies of the state-of-the-art were investigated. Regarding verification and validation techniques of critical software, we can consider different approaches:

- The first approach, based on formal proof, has been proved to be unsuitable in the case of very complex systems (explosion of proof algorithms) or the formal expression of which is inappropriate (lower layers for example).
- The second approach based on simulation is limited by the amount of test cases that have to be generated in order to cover all possible cases.
- The third approach consists in testing of the system in real conditions. However, reaching the validation objectives in terms of testing (hundreds of millions of kilometres) is tedious or even infeasible considering the lifecycles of these systems. Furthermore, the definition and realization of test drives campaigns is proving difficult to be representative and comprehensive.

There are currently attempts to resolve these problems, for example by trying to create tool chains based on bricks such as Matlab / Simulink and / or Statemate and / or SCADE + DesignVerifier and / or Prover and / or MaTeLo and / or Teststand ... But there is today no integrated solution that addresses all of the issues raised by COVADEC (i. e. including incorporating a detailed analysis of scenarios and detection component).

The new approaches developed by COVADEC project, will give the opportunities to propose different ways to assess ADAS function and to establish safety objectives compatible with camera based systems.

In a previous paper [UCAAT 2014], we focused primarily on the COVADEC tool chain. In this paper, the main subject is to describe the process developed in order to create an efficient test database from a test model.

4 Methodology and technological locks

Today, there is no standard method taking into account the constraints related to the use of ADAS based on ADAS sensors such as cameras.

4.1 Using a statistical approach

Statistical tests as currently proposed by MaTeLo should be immersed in the ADAS context. They must be mixed with the potential of test benches and simulators environment and adapted to the analysis of the automotive dependability.

To carry out the design and validation stages for object detection systems based on cameras, numerous driving hours are required to be processed. Some cases that can be defined as critical may rarely appear or never appear during this driving campaign. We will use simulation to cover such cases. The simulation must produce synthetic image data as close as possible to reality so that the evaluated algorithms have a behaviour identical to in similar real driving conditions.

The statistical approach will also allow to address the sensitivity of the system behaviour in case of small changes of the vehicle environment. It has to be underlined, though, that many parameters defining the vehicle environment are non-independent. As a direct consequence, the values of various parameters that are exploited for the generation of tests on a statistical basis cannot be drawn completely independently. For example, the number of other vehicles (traffic density) and driver behaviours are not independent of the type of road. If the parameters are not independent, random selection may lead to generate situations that do not exist in reality, and also misrepresent the likelihood of certain situations.

The statistical approach to test generation should be supplemented with two difficulties:

- The first difficulty is a practical one: we must exhaustively know the parameters incompatibility matrix. If a driving situation can be characterized by dozens or even hundreds of parameters, knowing this matrix is not trivial.
- The second one is theoretical: the Monte Carlo method assumes that the parameters are independent, and the desired probability distribution specifically corresponds to this case. It will therefore be necessary to consider how to correct the Monte Carlo method to account for the dependence of the parameters before selecting the relevant test cases.

4.2 Running the test cases in the ad hoc environment

Once test cases have been identified and generated, it is necessary to be able to run them in an automated way to manage a large number of test cases. It is also necessary to reduce the time required for their implementation through the use of high-performance parallelized computing. Today there are such tools, in particular for the management of test benches, HIL (Hardware In the Loop) systems, but no one incorporating the tools dedicated to ADAS architectures.

The challenge is to provide a tool that is both easily accessible in terms of user interface and configuration (import test cases, accessibility of execution reports), modular (able to accommodate execution targets like RTMaps and Pro-SiVIC but extendable thereafter to other environments, such as Simulink) and high performance (execution distributed on multiple machines, no duplication of software resources to handle such as sensor data or 3D records that are particularly large).

4.3 Being able to evaluate the test results with an oracle

Another issue is to build a usable oracle, automatic and that takes into account the wide variability of situations. A secondary challenge is to determine the best location for the implementation of this oracle in the I-DEEP platform.

4.4 Traceability of requirements

In COVADEC the test methodology targets the verification and validation of the considered ADAS in terms of availability, reliability and functional safety. These main requirements are expressed in a restricted set of requirements, prescribing the targeted objectives of the system as rates of availability over time or reliability on detections. Hence the fulfilment of this kind of requirements can only be evaluated by taking into account the integral test campaign and the traceability of these requirements to the test sequences has no vocation to be managed in a refined manner.

On the other hand, some requirements express some behavioural rules of the ADAS when confronted to identifiable environment perturbations (e.g. inhibition rules). Then the traceability of this kind of requirements to the corresponding test sequences shall be exploited in order to provide additional information and coverage metrics for global test campaign analysis.

5 Test cases automatic generation

5.1 Problematic

ADAS validation is a complex issue, particularly for camera based systems, because these functions maybe facing a very high number of situations that can be considered as infinite. But some situations will have more influence than others on the response of the ADAS function and some will occur more frequently. So, although all situations cannot be covered by test, it is possible to reduce the space to be tested in an area that can be small enough to make test possible by choosing to test the most representative and the most influential situations that an ADAS can encounter.

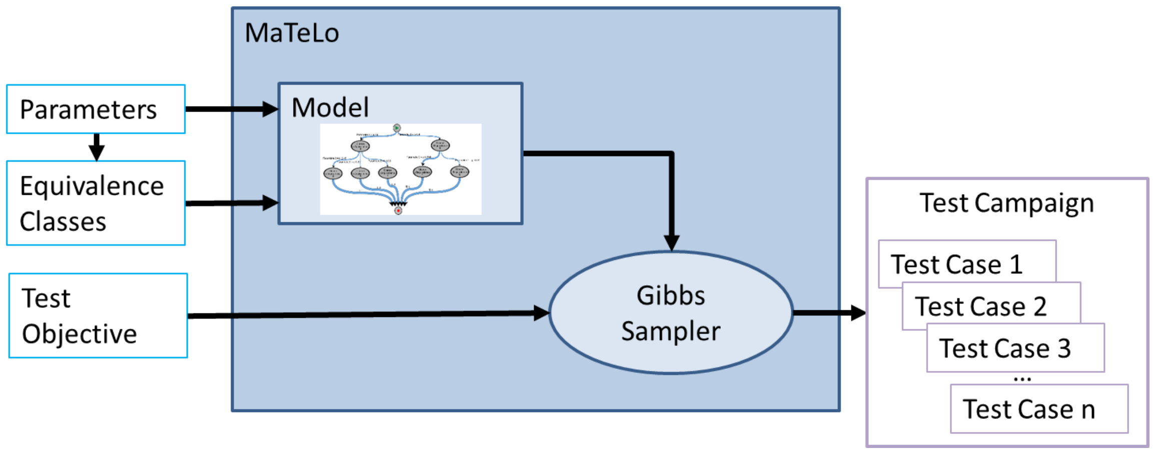
Whatever the nature of data used for validation, real or simulated, the Model-Based Testing (MBT) approach can be used to automatically build a complete test database which meets these objectives of limited size while covering most of the situations that most influence the ADAS function under test.

5.2 Model Based Testing (MBT)

The tool used for MBT is MaTeLo (Markov Test Logic). MaTeLo is an MBT tool, which makes it possible to build a model of the expected behaviour of the system under test (SUT) and then to generate, from this model, a set of test cases suitable for particular needs (for instance, testing only the most frequently used functions of the system, or having 100% coverage of system requirements). MaTeLo is based on Markov chains. For non-deterministic generation of test cases, MaTeLo uses the Monte Carlo methods, associated with generation strategies adapted to user needs. To cope with the combinatorial explosion, we couple the graph generated by MaTeLo to an ad hoc *random scan Gibbs sampler* (RSGS), which converges at geometric speed to the target distribution as explained later in the paper. Thanks to these test acceleration techniques, MaTeLo also makes it possible to obtain a maximal coverage of system validation by using a minimum number of test cases. As a consequence, the number of driving kilometres needed to validate an ADAS is reduced.

5.3 Summary of the test cases generation

The following figure gives a summary of the test cases generation:



Further details are given in the following parts.

5.4 Global strategy

Test case generation as proposed in the MaTeLo tool faces the question of inherent combinatorial explosion. Typically, the problem is to produce samples of large random vectors, the components of which are possibly dependent and take a finite number of values with some given probabilities. One important constraint is to generate almost all situations in the most economical way. In general this task can be considered from two points of view: deterministic (via binary search trees) or stochastic, via Markov chain Monte Carlo (MCMC) sampling. In the COVADEC project, we choose the probabilistic approach, which will rely on the implementation of a Gibbs sampler, briefly described below.

- In a first step, starting from the simulation graph generated by MaTeLo, the idea is to construct a Markov random field (see section 6.4). When the parameters are locally dependent, this can be achieved by means of Bayes' formulas.
- Then test cases will be obtained by implementing Gibbs samplers. In particular, we shall strive to optimize the convergence rate toward equilibrium, since it is known from the theory that the speed of convergence is exponentially fast.

5.5 Gibbs samplers and random fields

In order to simulate systems with large state-space and given multi-dimensional distributions, such as those encountered in statistical physics to study equilibrium properties, powerful methods have been proposed as soon as in the 1950's. In particular, the Metropolis-Hastings's algorithms [MET], [HAST]. In the context of image processing, where digitized images can be viewed as the realization of some random field, one must quote the seminal Gibbs sampler work [GEM].

5.5.1 Markov Random Fields (MRF)

For an introduction to the properties of the mathematical objects presented below, the reader is referred to e.g. [GRI], [BRE].

Let *V* denote the number of significant parameters in the system. We want to simulate the random vector $X = (X_1, X_2, \dots X_V)$, where each component $\mathrm{X_i}$ takes its values in a finite space Λ_i, usually called the phase space, with $|\Lambda_i| = C_i$. Typically $0 < C_i \approx 10$, and $V \approx 10^2$. The variables X_i are in general dependent. Thus a configuration $x = (x_1, x_2 \dots x_V)$ written with lowercase letters belongs to the space $\Lambda = \prod_{i=1}^{i=V} \Lambda_i$.

Of special interest will be MRF satisfying local interaction properties. This classical notion relies mainly on conditional expectation, after having defined a convenient topology on the set of indices $S = \{1,2, \dots V\}$ of the components of X, which from now on will be rather called the set of sites. Then one can define a neighbourhood system on S (i.e. a topology), which is a family $F = \{N_{s \in S}\}$ such that, $\forall s \in S$,

$$s \notin N_s \text{ and } t \in N_s \Rightarrow s \in N_t.$$

The subset $\mathcal{N}_s$ is the neighbourhood of the site s. In a more general graph framework, S is the set of vertices and F defines the edges: s and t are linked by an edge if and only if they are neighbours, i.e. $t \in N_s$.

Definition 1. The random field X is called a Markov random field with respect to the neighbourhood system F if for all sites $s \in S$ the random variables X_s and $(X_i, i \notin N_s)$, are independent given the $(\mathrm{X_i}, \mathrm{i} \in \mathcal{N}_{\mathrm{s}})$.

Let $\pi\,(.)$ denote the multivariate probability measure of the vector X, so that $\pi(x) \overset{\text{def}}{=} P(X = x)$. Then π is a *Gibbs distribution* relative to the graph $\{S, F\}$ if it is of the form

$$\pi(x) = \frac{1}{Z_T} e^{-\frac{U(x)}{T}},$$

where $T > 0$ is the temperature, $U(x)$ is the energy of the configuration x, which derives from some potential, and Z_T is the normalizing constant. Under the so-called *positivity condition* (Brook's Lemma, which is in particular satisfied when $\pi(x) > 0, \forall x \in \Lambda$), an important theorem due to Hammersley and Clifford shows the equivalence between Gibbs distributions and MRF, which in fact are essentially the same objects.

5.5.2 Gibbs samplers (GS)

Gibbs sampling has numerous applications and became one of the most popular routine amongst MCMC simulation methods. It applies to any multivariate distribution of the form $\pi(x_1, x_2 \ldots x_V)$. There are two main families of GS: Random scan Gibbs samplers and Periodic Gibbs samplers.

Random scan Gibbs sampler (RSGS)

The principle is simple: at each step, one selects at random a site (coordinate) $s \in S$, and then compute the new value y_s of the corresponding site according to the conditional probability

$$\pi(y_s | \, x_j \, j \neq s) = \pi(y_s | \, x_j \, , j \in N_s).$$

Let α_s denote the probability of visiting the site s, with $0 < \alpha_s < 1$ and $\sum_1^V \alpha_s = 1$. The algorithm does construct a Markov chain $\{X(t), t = 0,1, \ldots\}$, the evolution of which is as follows.

(a) Select an initial vector $X(0)$ and a probability vector $(\alpha_1, \alpha_2, \ldots, \alpha_V)$.
(b) On the t-th iteration,
- Choose an index *s* with probability α_s;
- Generate $X_s(t)$ with probability $\pi(X_s \mid X_j \, (t-1), j \in N_s)$;

(c) Repeat step (b) until reaching equilibrium.

It can be shown that the Markov chain $X(t)$ is reversible, so that its invariant measure is precisely the distribution π of the vector X.

Periodic Gibbs sampler

Here sites are not chosen at random, but in well-determined order fixed in advance, say $(s_1, s_2, \ldots, s_V)$ which is a permutation of $(1,2, \ldots, V)$. The algorithm generates a Markov chain $Z(t)$ as follows. One first draws X_{s_1} conditoned on the current state of the other sites, then draw X_{s_2} in the same way, etc., until X_{s_V}. After this sweep, one says that the Markov chain $Z(t)$ has moved exactly one step and it is not difficult to show that π is its invariant measure.

Speed of convergence

As a consequence of standard results on Markov chains, the speed of convergence to the equilibrium of the Gibbs samplers is geometric. This means that we have (see for example [BRE]) :

$$|X(0)\mathbb{P}^n - \pi| \leq \frac{1}{2}|X(0) - \pi|\delta(\mathbb{P})^n,$$

where $\mathbb{P}$ stands for the stochastic transition matrix of the Markov chain obtained from a Gibbs sampler, and $0 \leq \delta(\mathbb{P}) \leq 1$ is the Dobrushin's ergodic coefficient of $\mathbb{P}$, with

$$\delta(\mathbb{P}) = 1 - \, inf_{\, i,j \in \Lambda} \sum_{k \in \Lambda} p_{ik} \wedge p_{kj} \, ,$$

the p_{ik} 's being the elements of $\mathbb{P}$.

Computing satisfactory explicit bounds for $\delta(\mathbb{P})$ is a difficult (mostly open) problem, which depends on the kind of GS considered. For some global theoretical results in this respect, one can see for instance [LIU] and [LEV].

Indeed, the rate of convergence depends deeply on the structure of the underlying MRF describing the system. In the COVADEC project, we shall implement a RSGS. Then, by using the specific properties of the graph produced by MaTeLo, we shall analyse the speed of convergence as a function of the free probability vector $(\alpha_1, \alpha_2, \ldots, \alpha_V)$ introduced above.

5.6 Parameters of the MaTeLo model

In order to cover up all the situations that the ADAS systems may face, it is necessary to provide a model of the environment and driving context. The objective is to provide a meta-model of the test sequences, taking into account influential parameters that express the variability of situations the system may encounter. The construction of such a model involves taking into account parameters of heterogeneous nature, with very diverse impacts on the scene as perceived by the system.

This model must gather information about the environment in which evolves the ADAS (landscape, road type, curvature, infrastructure, etc.), driving situations (behaviour of the equipped vehicle and surrounding vehicles), weather conditions (sun, rain, fog, etc.) and known troublemakers.

The modelling of the environment needs to be as comprehensive as possible. Indeed, the model is supposed to represent any circumstances that the vehicle may encounter. Therefore, if an actually influencing parameter is forgotten, it will not be considered when creating test cases and thus the simulator may possibly never generate the situation corresponding to disturbing values for this parameter. However, these situations are potentially present in the actual video databases, even if the influential parameter in question is not explained.

We have defined several categories of influential parameters, shown below. These categories include parameters according to their nature and permit any transposition to another function.

5.6.1 Weather conditions

Weather conditions have an impact on how the ADAS will perceive a scene. This includes not only the weather as such, but also disturbances induced by these conditions as well as the lighting conditions of the scene.

5.6.2 Structure of the road and of the environment

This category includes the intrinsic characteristics of the road, that is to say the parameters to accurately describe its structure (curvature, topology, number of lanes, etc.), as well as its appearance and overall look (surface, marking, etc.).

5.6.3 Behaviour of the equipped vehicle

This category is used to express the behaviour of the equipped vehicle in a test sequence, both in terms of speed or trajectory rate of change. In addition, this category includes the actions of the driver that may impact the function without implying a change in the trajectory or speed (e.g. wiper operation).

5.6.4 Behaviour of surrounding vehicles

The presence of other vehicles can influence the perception of the scene by the ADAS either as a target vehicle or as a barrier masking what the ADAS should detect. The behaviour of other vehicles is described by a set of parameters identical to those defined for the behaviour of the equipped vehicle for which we have added parameters relating to their positioning in the scene as well as changes of trajectories they can make.

5.6.5 Pedestrians

This category of parameters can express how pedestrians will evolve in the scene (number, trajectory, crossing the road, etc.).

5.6.6 Obstacles and disturbances

This category includes all the obstacles and other disturbances known to have an impact on how the ADAS will perceive a driving situation. We grouped the barriers in several sub-categories, namely:

- Fixed Targets set on the way: this includes work pads, a stationary vehicle, a lost loading or any other object that may be on the way.
- Barriers at the trajectory limit: this includes road signs, guard rails, or a stationary vehicle.
- Pedestrians in particular situations

5.6.7 Equivalence classes

The range of possible values for each parameter is divided into several equivalence classes, for two main reasons:

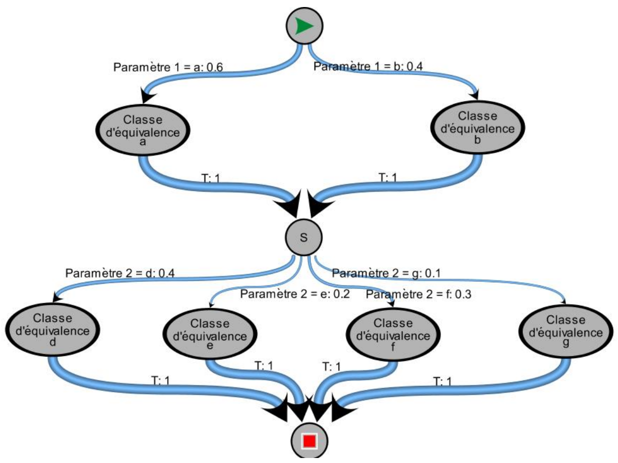
- To select sets of values having a real impact on the ADAS function. This corresponds to the notion of "range", all situations are assumed equal within the range (e.g. 130 km / h and 131 km / h are considered equivalent in terms of ADAS, but 20 km / h belongs to a different equivalence class).
- To manage the dependencies between parameters. Indeed, some values of an influential parameter X may not be possible or have a different probability if the parameter has a value Y (Y of X correlation - examples: "night" and "sunny" are incompatible; "speed> 130 km / h" and "urban environment" is an unlikely event).

When building test campaigns, that is to say, sets of test cases which will be run for the ADAS function, if one test case has all its values in exactly the same equivalence classes another test case, it will be considered duplicate and eliminated from the campaign.

5.7 Structure of the MaTeLo model

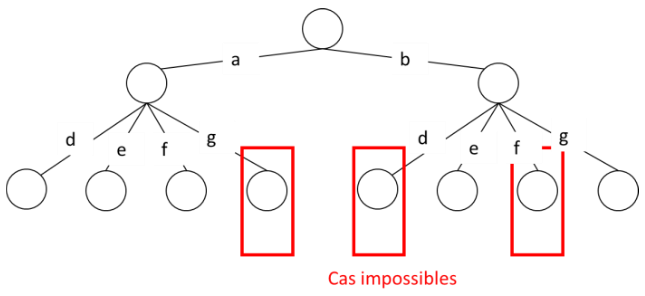
The structure of the MaTeLo model to generate test cases is based on the influential parameters. In particular, the dependence between parameters is modelled as a series of dependent transitions in the MaTeLo model.

Indeed, if the parameters were independent of each other, the most natural way to build a MaTeLo model would be to create a single chain as follows:

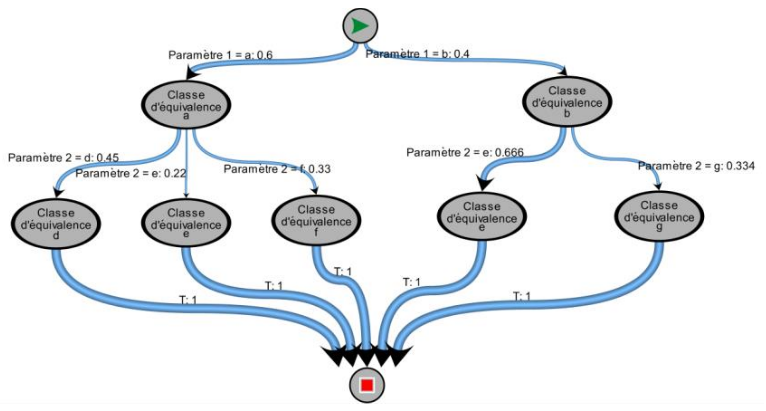


This model in the case of dependent parameters is no longer acceptable, since such model generates test cases that cannot occur in reality, distorting the representativeness of generated test campaigns.

The following graph is the graph of all possible cases of the above MaTeLo model. Let suppose that the case identified by red rectangles are impossible cases.



It is therefore proposed to build the MaTeLo chains as illustrated by the following example:



The modelled dependencies can be seen in two ways:

- Either in terms of reachable equivalence classes,
- Or in terms of probability of choosing an equivalence class knowing the equivalence class chosen for the previous parameter.

A MaTeLo model is a directed graph, so there is a notion of order to draw parameters. As a result, the dependence between parameters constrains how to build the model, including the order in which they appear.

When the level of dependencies between parameters is low (interdependencies for up to three parameters), the MaTeLo models can be simplified by creating macro-parameters, which can transform a chain of dependencies in independent chains. For instance, the moment of the day (day or night) is linked to the apparent brightness of the scene, since the brightness is darker during the night than during the day. But we could consider only the brightness, which become a macro-parameter, with a bigger range of value than considering separately either the brightness of the day or the brightness of the night. To do this, it is necessary to calculate the probabilities of the corresponding transitions, which can be obtained from the conditional probabilities that were on the initial model. Studies in the project have shown that the conditional probabilities linking the dependent parameters can be calculated from a MaTeLo model according to a simple algorithm, using Bayes formulas.

In conclusion it can be said that solutions already obtained in the project will meet satisfactory (although partially for now) to the problem of parameters dependency.

5.8 Parameters interpretation by the simulator

Influential parameters do not necessarily have direct translation in the simulator. A collaboration between ALL4TEC and ESI is therefore necessary to ensure that the parameters provided by MaTeLo can be properly applied in the simulator. They can be expressed in different ways in the simulator:

- thanks to existing simulation components (object models),
- through static resources called by these components (files),
- via the configuration of these components (script commands).

For performance practical aspects (related to loading times and simple definition of scenarios in the simulator), it is preferable to minimize the number of components creations and resource loads. Therefore, it is possible to create some components masking components creation or resource loading, and giving a more direct correspondence between influential parameters and script commands sufficient to describe them. The disadvantage of this approach is the need for creating a numerous components very specific to some subsets of scenarios.

6 Using both simulation and real data

The innovation is to manage data collection, using a statistical model based on MaTeLo tool. Instead of driving millions of kilometres aiming to encounter all function life situations, we are exploring within this project, the solution of reducing the number of kilometres by targeting the most influent conditions on the system.

A first work will be to collect sufficient real data to validate in depth the two COVADEC ADAS functions. However, in the case of some validation tests generated by MaTeLo, it is possible that the database does not cover these tests (taking into account all the various parameters / variables for this test / difficulty to test dangerous situations in real). A simulation tool must come to fill this gap in the database by synthesizing realistic sensor data for these test cases. Ideally it would be desirable that the simulation platform generates scenes, scenarios and sensor data just from the definition of the tests. In practice, certain steps must be performed in a preliminary phase before the execution of tests, in particular respective to the environment. The optimization of this process and the opportunities of generation for these items when running tests were examined in the project.

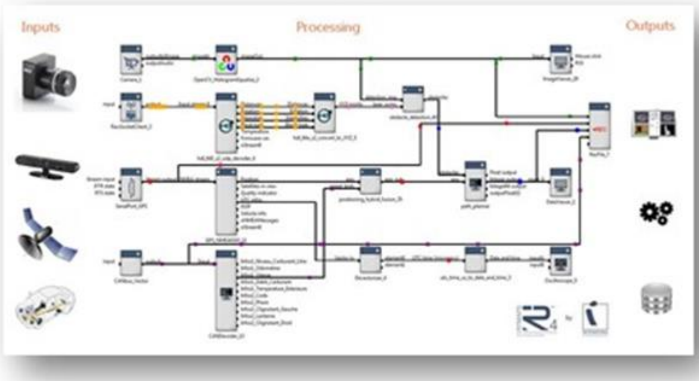
7 Testing tool chain for simulation

Once the test cases have been defined, it is necessary to inject them in a testing tool chain in order to execute them on the ADAS under validation. This tool chain is composed of:

- A simulator of scenarios, environments and video camera sequences (Pro-SiVIC – ESI)
- High performance ADAS data recorders (Intempora dataloggers)
- A ground-truth extraction tool for recorded data (IBEO Evaluation Suite) (VALEO)
- A framework for virtual-time or real-time execution of ADAS algorithms (RT-MAPS – Intempora)
- An ADAS hardware architecture simulator (Rabbits – TIMA)
- A test execution automation server (I-DEEP – Intempora)

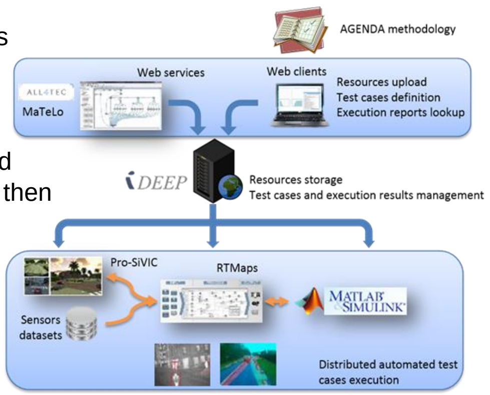
Pro-SiVIC is a simulation software environment specialized in the advanced rendering of ADAS sensors (cameras, lidars, radars, GPS, communication systems…). It offers complex sensor models as well as environments taking into account numerous physical and electronic characteristics (for a camera for instance, the point is to model distortion, noise, atmospheric and climatic conditions, lighting conditions…). The key aspect for the validation process is the ability to control the characteristics of the environmental conditions. Pro-SiVIC also allows to integrate vehicle dynamic models, to setup complex driving situations in complete environments, objects animation (such as pedestrians for example). Pro-SiVIC can operate in real time or virtual time which allows addressing tests and validation use cases of ADAS functions with or without human or ECU in the loop.

RTMaps is a modular (component-based) software framework for rapid development and optimized execution of real-time applications having to manage, process and fuse numerous high-bandwidth, asynchronous and heterogeneous, sensors data streams such as cameras, lidars, radars, CAN bus, GPS, IMUs, V2V and V2I communications, etc.). RTMaps also offers data recording of any kind of ADAS sensors, then synchronized playback, in real-time or virtual time, in order to allow offline developments for perception, data fusion, communications, decision making and command-control (developments, tests, validation and benchmarking). RTMaps can also be connected to simulation and/or command control tools such as Pro-SIVIC.



I-DEEP is a test execution automation server dedicated to validation of perception and decision making function for ADAS, particularly functions based on vision.

- I-DEEP can store recorded sensors datasets and their associated ground-truth datasets and/or simulation scenarios resources, it can as well host image processing / signal processing / data fusion algorithms to be tested (as integrated into RTMaps plugins), and then



allows to define and execute automatically the numerous test cases on cluster of calculators.

- I-DEEP also offers a dual approach for validation of ADAS functions making use of simulation on the one hand and real datasets playback on the other hand. These two approaches are very complementary, simulation offering a comprehensive control of the scenario and its environmental conditions as well as the capability to test dangerous situations, whereas taking advantage of real data playback capabilities allow extension of the tests under maximum realism conditions.

Rabbits is a fast hardware/software simulator capable of co-simulating multiprocessor systems on chip. It leverages on QEMU for processor modelling, and SystemC TLM for hardware IPs modelling. Rabbits supports many parameters, such as variable number of processors, memory size, cache availability, cache size, support of specific instructions, e.g. SIMD or floating point, etc. It also provides hardware IPs, such as memories, interrupt controller, uart, frame buffer, etc. Even though being fairly abstract, the simulation technology allows to get timing evaluation of the hardware/software system, though high level instruction execution time, instruction and data cache models, and interconnect models. In this project, Rabbits is used as a simulator of the ADAS hardware architecture with the aim of doing design space exploration of both the hardware platform and the software implementation. We did two parallel implementations of a line departure warning algorithm in C. The first one uses coarse grain (i.e. thread level) parallelism, and is executed on platforms that embed from 1 to 8 Cortex A9, leading to a factor of acceleration of 3 on the 8 core platform as compared to the unicore platform. The second implementation uses the SIMD extensions of the NEON coprocessor to express instruction level parallelism. Thanks to its capability of performing highly parallel instructions, we gained a factor of two on the already accelerated coarse grain implementation. Rabbits has been inserted in the whole design flow as an RTMaps component when targeting the validation of an optimized implementation. RTMaps pre-processes the images generated by Pro-SiVIC and sends them, though an I-DEEP interface, to Rabbits. Rabbits is concurrently running the cross-compiled LDW software that reads the images through a fake camera device hooked on the I-DEEP interface, performs the computations on them, and reports to RTMaps, through a fake serial interface also hooked on an I-DEEP interface, the status of the car on the road. RTMaps then feeds the rest of the processing chain with this information so that the appropriate decision can be taken by the system.

8 Expected benefits and major results

We expect benefits at many levels:

- Enhance the global knowledge of ISO 26262 applicability to design and validation of ADAS sensors, and shed light on its limitations, in order to propose solutions.
- Reduce the number of kilometres for validation of ADAS, by using a statistical model and by optimizing test plans using ‘equivalence classes’ principle.
- Build an ADAS validation platform (model in the Loop and software in the loop) combining real and simulation environment data.

At this stage of the project, the methodology has been entirely developed and tested on small samples of the problematic. Further developments currently in progress concern improvement of the simulator Pro-SIVIC (in order to manage a wide range of elements in the videos), of the test automation server I-DEEP (in order to use the real data from driving campaigns for tests) and the implementation of Gibbs samplers algorithms in the test case generator (MaTeLo).

Currently, a first series of tests has demonstrated a reduction in the required testing effort (considering the safety goals) by almost 90%, compared with the other available validation methods. This effort reduction target should be confirmed during the full-scale validation of the two ADAS functions expected to start from February 2016.

9 Bibliography

[BRE] P. Brémaud (1995), *Markov Chains*, Springer.
[GEM] S. & D. Geman (1984), IEEE Trans. on Pattern Analysis and Machine Intelligence, 6, 721-741.
[GRI] G. Grimmett (2010*), Probability on Graphs*, CUP.
[HAST] Hastings (1970), Biometrika, 57 (1), 97-109.
[LEV] R.A. Levine & al. (2006), Journal of Multivariate Analysis, 97, 2071-2100.

[LIU] J.S. Liu & al. (1995), Journal of the Royal Stat. Society, Series B, 57 (1), 157-169.
[MET] Metropolis & al., 1953, The Journal of Chemical Physics 21, 1087-1092.
[UCAAT] L. Raffaelli & X. Rouah (2014), Model-Based Testing and Test Automation applied to Advanced Driver Assistance Systems Validation.

Session 16

Safety & Security

Thursday 28th, 11:45 – Ariane 2

What's Security Level got to do with Safety Integrity Level?

Jens Braband

Siemens AG, Braunschweig, Germany
jens.braband@siemens.com

Abstract. Some recent incidents and analyses have indicated that possibly the vulnerability of IT systems in railway automation has been underestimated so far. Due to several trends, such as the use of commercial IT and communication systems or privatization, the threat potential has increased. This paper discusses the relationship of IT security and functional safety from the perspective of their integrity measures.

Keywords. Railway, IT Security, Safety, Threats, IT Security Requirements, Security Level, Safety Integrity Level.

1 Introduction

Recently, reports on IT security incidents related to railways have increased as well as public awareness. For example, it was reported that on December 1, 2011, "hackers, possibly from abroad, executed an attack on a Northwest rail company's computers that disrupted railway signals for two days" [1]. Although the details of the attack and also its consequences remain unclear, this episode clearly shows the threats to which railways are exposed when they rely on modern commercial-off-the-shelf (COTS) communication and computing technology. However, in most cases, the attacks are denial-of-service attacks leading to service interruptions, but so far not to safety-critical incidents. Many other attacks that have been reported or have been claimed to be possible, could fortunately be shown to be unfounded or were oriented towards public relation, e. g. a hack of Nuremberg's automated metro was performed on an unprotected self-made system [2]. However, in 2014, the German Federal Agency for IT Security (BSI) reported the first successful attack on critical industrial infrastructure. As a consequence a blast furnace was damaged and had to be shut down [3].

What distinguishes railway systems from many critical infrastructures is their inherent distributed and networked nature with tens of thousands of track-kilometers for major operators, or even more. Thus, it is not economical to completely protect against physical access to this infrastructure and, as a consequence, railways are very vulnerable to physical denial-of-service attacks leading to service interruptions.

Another distinguishing feature of railways from other systems is the long lifetime of their systems and components. Current contracts usually demand support for at least 25 years and history has shown that many systems, e.g. mechanical or relay interlockings, last much longer. IT security analyses have to take into account such long lifespans. Some of the technical problems are not railway-specific, but are shared by a few other sectors such as Air Traffic Management.

Publications and presentations related to IT security in the railway domain are increasing. Some are particularly targeted at the use of public networks such as Ethernet or GSM for railway purposes, while others directly pose the question "Could rail signals be hacked to cause crashes?"[4]. While in railway automation harmonized functional safety standards were elaborated more than a decade ago, up to now no harmonized international IT security requirements for railway automation exist.

This paper starts with a discussion of the normative background and then discusses the similarities and dissimilarities of IT security and functional safety, in particular from the point of view of their integrity measure Security Level (SL) and Safety Integrity Level (SIL), respectively. In particular the requirements for SL and SIL are compared, e. g. which SL can be covered by SIL.

2 Normative background

In railway automation, an established standard for safety-related communication, EN 50159 [5] exists. The first version of the standard was elaborated in 2001. It has proven quite successful and is also used in other application areas, e.g. industrial automation. This standard defines threats and countermeasures to ensure safe communication in railway systems. The methods described in the standard are partially able to also protect railway system from intentional attacks, but not completely. Until now, additional organizational and technical measures have been implemented in railway systems, such as separated networks, etc., to achieve a sufficient level of protection.

The functional safety aspects of electronic hardware are covered by EN 50129 [6]. However, IT security issues are taken into account by EN 50129 only as far as they affect safety issues, but, for example, denial-of-service attacks often do not fall into this category. Questions such as intrusion protection are only covered by a single requirement. However, EN 50129 provides a structure for a safety case which explicitly includes a subsection on protection against unauthorized access (both physical and informational), so it is already a “security-informed safety case”. Other security objectives could also be described in that structure.

On the other hand, industrial standards on information security exist. ISO/IEC 15408 [7] provides evaluation criteria for IT security, the so-called Common Criteria. This standard is solely centered on information systems and has, of course, no direct relation to safety systems. IEC 62443 [8], also known as ISA 99, is a set of 12 standards currently elaborated by the Industrial Automation and Control System Security Committee of the International Society for Automation (ISA). This standard is not railway-specific and focuses on industrial control systems. It is dedicated to different hierarchical levels, starting from concepts and going down to components of control systems.

How can the gap between information security standards for general systems and railways be bridged? The bridge is provided by the European Commission Regulation on Common Safety Methods No. 402/2013 [9]. This Commission Regulation mentions three different methods to demonstrate that a railway system is sufficiently safe:

a) by following existing rules and standards (application of codes of practice),
b) by similarity analysis, i.e. showing that the given (railway) system is equivalent to an existing and used one,
c) by explicit risk analysis, where risk is assessed explicitly and shown to be acceptable.

We assume that, from the process point of view, IT security can be treated just like functional safety, meaning that threats would be treated as particular hazards. Using the approach specified under a) IT security standards may be used in railway systems, but a particular tailoring would have to be performed due to different safety requirements and application conditions. With this approach, a code of practice that is approved in other areas of technology and provides a sufficient level of IT security can be adapted to railways. This ensures a sufficient level of safety.

However, application of the general standards requires tailoring them to the specific needs of a railway system. This is necessary to cover the specific threats associated with railway systems and possible accidents and to take into account specific other risk-reducing measures already present in railway systems, such as the use of specifically trained personnel.

As a basis of our work, the IEC 62443 [8] has been selected, as this standard series seemed to provide the best fit. With this approach, a normative base has been developed by the German standardization committee DKE [10], based on IEC 62443 tailored for railways, considering railway-specific threats and scenarios and yielding a set of IT security requirements. Assessment and certification of such a system can be carried out by independent expert organizations. Safety approval in Germany could then be achieved via the governmental organizations Federal German Railways Office (Eisenbahn-Bundesamt, EBA) for railway aspects and Federal German Office for Security in Information Technology (Bundesamt für Sicherheit in der Informationstechnik, BSI) for IT security aspects.

3 Basic concepts of IEC 62443

A total of 12 standards or technical specifications is planned in the IEC 62443 series of standards that cover the topic of IT security for automation and control systems for industrial installations entirely and independently. This series of standards adds the topic of IT security to IEC 61508 which is the generic safety standard for programmable control systems. Up to now, though, IEC 61508 and IEC 62443 have only been loosely linked.

IEC 62443 addresses four different aspects or levels of IT security:

– general aspects such as concepts, terminology and metrics: IEC 62443-1-x
– IT security management: IEC 62443-2-x
– system level: IEC 62443-3-x
– component level: IEC 62443-4-x

Today, however, the parts of IEC 62443 are still at different draft stages. Only a small number of parts such as IEC 62443-3-3 have already been issued as an International Standard (IS) or a Technical Specification (TS). Due to the novelty of the IEC 62443 series in this section, the essential concepts of IEC 62443 will be explained briefly so as to improve understanding of the adaptation and embedding of IT security in compliance with IEC 62443 into EN 50129.

3.1 IT security management

An IT security management system (ISMS) shall be established for operation of the system. The aim of an ISMS is to continuously control, monitor, maintain and, wherever necessary, improve IT security. In the case of the ISMS, IEC 62443 is based on the general stipulations of the ISO/IEC 17799 and ISO/IEC 27000 series. It details these general standards by adding specific aspects for safety-related control systems. If an ISMS is already established, it may remain in use. However, the essential principles of the ISMS according to IEC 62443 should be introduced or integrated. In the event of integration into an existing ISMS, the special technical aspects of a safety-related railway system shall be observed. Due to the specific framework, unreflected adoption of the stipulations from IT security does not make sense and in most cases can only be implemented with difficulty. The DKE standard [10] offers a comparison of IT security elements from common standards, and is intended to assist integration.

One key task of ISMS is risk management. This includes the consideration of all functional components of the system together with those that are specific to IT security.

3.2 System definition

The system and its architecture are divided into zones and conduits. The same IT security requirements apply within each zone. Every object, e.g. hardware, software or operator (e.g. administrator) shall be assigned to precisely one zone and all connections of a zone shall be identified. A zone can be defined both logically and physically. This approach matches the previous approach for railway signalling systems very well, as has been used as the basis in numerous applications [11]. Figure 1 shows a simple application of the concept, the connection of two safety zones by a virtual private network (VPN) connection as the conduit.

The conduit would consist of the gateways at its borders and the connection in between whatever the actual network would look like. Strictly speaking management would itself be a zone with conduits connecting it with the gateways.

This example may serve as a blueprint for the connection of zones with similar IT security requirements. If zones with different IT security requirements shall be connected, different types of conduits, e. g. one-way connections or filters have to be applied.

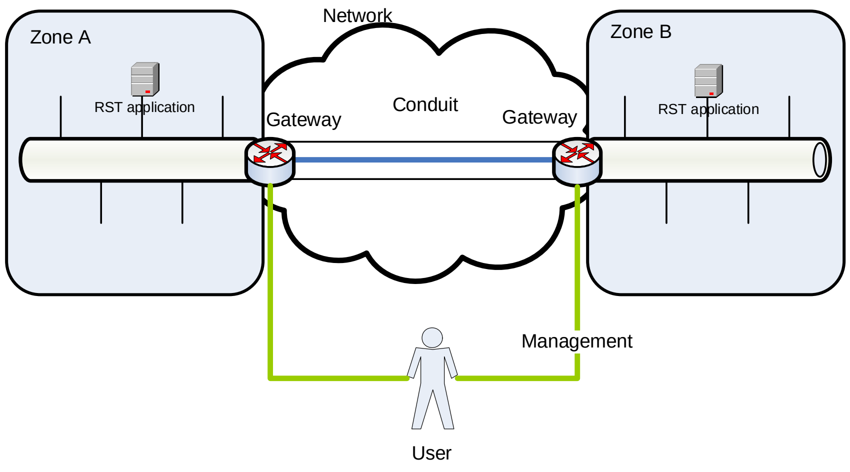


Figure 1: Zone and conduit architecture example

3.3 IT security requirements

In IEC 62443, the IT security requirements are grouped into 7 fundamental requirements:

1. identification and authentication control (IAC)
2. use control (UC)
3. system integrity (SI)
4. data confidentiality (DC)
5. restricted data flow (RDF)
6. timely response to events (TRE)
7. resource availability (RA)

Normally, only the issues of integrity, availability and data confidentiality are considered in IT security. However, the fundamental requirements IAC, UC, SI and TRE can be mapped to integrity, RA to availability and DC and RDF to confidentiality. Instead of defining a seven level Evaluation Assurance Level (EAL) as in the Common Criteria, which is to be applied with regard to the IT security requirements, a four stage IT security requirement level is defined. A possible explanation might be that also most safety standards define four levels. But it would lead to quite demanding and sometimes unnecessary requirements if the level would be the same for each of the foundational requirements. For example confidentiality often plays a minor role for safety systems and encryption of all data might lead to complications in testing or maintenance of safety systems. So different levels may be assigned for each of the seven foundational requirements. The SL values for all seven basic areas are then combined in a vector, called the SL vector. Note that this leads theoretically to 16384 possible different SL.

The SL are defined generically in relation to the attacker type against whom they are to offer protection:

SL 1	Protection against casual or coincidental violation
SL 2	Protection against intentional violation using simple means with few resources, generic skills and a low degree of motivation
SL 3	Protection against intentional violation using sophisticated means with moderate resources, IACS-specific skills and a moderate degree of motivation
SL 4	Protection against intentional violation using sophisticated means with extended resources, IACS-specific skills and a high degree of motivation

Sometimes a SL 0 (No protection) is also defined, but as we argue below, at least for safety-related systems this is not an option and so we do not discuss SL 0 further in this paper.

For one zone, for example, (4, 2, 3, 1, 2, 3, 2) could be defined as an SL vector. Once its vector is defined, IEC 62443-3-3 calls for a complete catalogue of standardised IT security requirements for the object under consideration, e.g. for a zone.

It is necessary to take into account the fact that IEC 62443 defines different types of SL vectors:

- The target SL (SL-T) is the SL vector that results as a requirement from the IT security risk analysis.
- Achieved SL (SL-A) is the SL vector which is actually achieved in the implementation when all the framework conditions in the specific system are taken into account.
- SL capability (SL-C) is the SL vector that the components or the system can reach if configured or integrated correctly, independent of the framework conditions in the specific system.

4 Relationship of SL and SIL

First, we should recall that, like IEC 61508, EN 50129 defines only four different Safety Integrity Levels (SIL). A SIL "indicates the required degree of confidence that a system will meet its specified safety functions with respect to systematic failures"[6]. Other target measures are defined with regard to random failures, but for IT security we are only concerned with systematic failure [12].

A first look at EN 50129 reveals that safety systems also have to deal with human errors and foreseeable misuse, which corresponds well to SL 1. For this reason SL 0 is not acceptable for safety-related systems. So we can conclude that for any safety system, even if IT security threats can be effectively ruled out, the basic IT security requirements SL 1=(1,1,1,1,1,1,1) should be fulfilled. So it is an interesting exercise to discuss the SL 1 requirements and evaluate whether these are normally fulfilled by safety systems developed according to EN 50129.

In a first step we take a more general look at the Foundational Requirements (FR). Due to functional safety criteria, not all requirement groups of IEC 62443 in applications for railway signalling systems have the same significance. Only the following requirement groups have direct relevance in the sense of functional safety:

1 unauthorised physical or logical access (IAC)
2 unauthorised use (UC)
3 manipulation of the system (SI)
6 response to events that is not timely (TRE)

For example, safety-related applications generally do not impose any requirements on the confidentiality of operational data. Therefore, apart from exceptions such as key management, further requirements for confidentiality can be discarded.

So in order to come up with a manageable number of SL vectors, we may as a first simplification and short hand notation set SL 1 as the default for all FR that are not directly safety-related. And we might work under the assumption that in a first approach all other FR may have the same importance. This would lead to four generic SL profiles: (1,1,1,1,1,1,1), (2,2,2,1,1,2,1), (3,3,3,1,1,3,1) and (4,4,4,1,1,4,1). It is admitted that additional SL profiles are necessary for particular zones or conduits. For example a zone containing a key management centre will deserve more demanding confidentiality requirements leading to another profile. But the idea would be to be able to cope with 5 to 10 profiles instead of 16384 possible combinations.

In a next step, we have discussed all 43 requirements from IEC 62443-3-3 in detail in order to find out, which are covered by EN 50129. According to the analysis in the annex many IT security requirements for SL1 are already adequately covered by railway safety standards or are not relevant to safety. These results are summarised in Table 1. They no longer need to be verified in each individual case for railway signalling applications.

Reference	Title	Assessment
SR 1.6	Management of wireless access processes	This requirement is not relevant for SL1.
SR 1.13	Access through untrustworthy networks	This requirement is not relevant for SL1.
SR 2.2	Use control in the case of radio connections	This requirement is not relevant for SL1.
SR 3.1	Communication integrity	This requirement is fulfilled by application of EN 50159.
SR 3.3	Verification of IT security functionality	This requirement is fulfilled by application of EN 50128.
SR 3.4	Software and information integrity	This requirement is fulfilled by application of EN 50128.
SR 3.5	Input validation	This requirement is fulfilled by application of EN 50129 and EN 50128.
SR 3.6	Deterministic output	This requirement is fulfilled by application of EN 50129 and EN 50128.
SR 4.1	Confidentiality of information	This requirement is not relevant for railway applications with SL1.
SR 4.3	Use encryption	This requirement is not relevant for railway applications with SL1.
SR 5.1	Network segmentation	This requirement is fulfilled by application of EN 50159.
SR 5.2	Protection of the zone boundary	This requirement is not relevant for SL1.
SR 5.3	Restriction of general communication between persons	Generally, voice communication is not part of the safety system. However, this requirement shall be exported to the operator.
SR 7.1	Protection against DoS attacks	This requirement is normally not contained in safety standards because it cannot be fulfilled by safety-related systems alone. The rule shall be exported to the operator.
SR 7.2	Resource management	This requirement is normally not contained in safety standards because it cannot be fulfilled by safety-related systems alone. The rule shall be exported to the operator.
SR 7.3	Backups of the automation system	This requirement is normally not contained in safety standards because it cannot be fulfilled by safety-related systems alone. The rule shall be exported.
SR 7.4	Restart and recovery of the automation system	This requirement is fulfilled by application of EN 50129.
SR 7.5	Emergency power supply	This requirement is normally not contained in safety standards because it cannot be fulfilled by safety-related systems alone. The rule shall be exported to the operator.
SR 7.6	Network and security settings	This requirement is fulfilled by application of EN 50128 and EN 50129.

Table 1 – IT security requirements that are already covered or are irrelevant

This means that for new safety-related systems it would be an advantage to implement all SL1 requirements from IEC 62443 (independent from the SIL) as most of them are already covered by safety standards (and some might not be relevant). In this case also an additional IT security certification for SL1 might be avoided as the requirements could adequately be included in the safety certification.

However a more detailed analysis (see the appendix) shows that starting with SL2 requirements there is no similar relationship with SIL anymore. The reason is that by definition the higher SL levels deal with intentional attacks which have only partially be covered by safety standards such as EN 50159 for communication. So also simple rules like “If you have a SIL x safety system then you must require at least SL x” cannot be justified as the allocation of SL depends also on the overall security architecture, e. g. physical protection, and not on the technical solution alone.

5 Summary

This paper has discussed the relation between SL from IEC 62443 and SIL from EN 50129 for safety systems. The major new results are:

- SL and SIL are completely different concepts, e. g. SL is a seven dimensional vector in contrast to the scalar SIL
- There is no simple relationship between SL and SIL
- SL 0 for safety-related systems is not acceptable. For safety systems, it is recommended to always take the requirements of SL 1 into account
- A preliminary proposal for SL profiles has been made in order to master the complexity of potentially 16384 SL vectors

Table 1 gives a summary of which requirements for SL 1 are already covered or not relevant from a safety perspective. The annex gives a more detailed discussion including a comparison with SL2 requirements.

The results should also hold for other related safety standards such as IEC 61508 as they build upon similar general principles, however the details would have to be checked and might differ.

6 References

1. Hackers manipulated railway computers, TSA memo says, http://www.nextgov.com/nextgov/ng_20120123_3491.php?oref=topstory, accessed on February, 7, 2012
2. Keine Hacker-Angriffe auf Nürnberger-U-Bahn, http://www.merkur.de/bayern/vag-gegen-vorwuerfe-keine-sicherheitsluecken-nuernberger-u-bahn-4654909.html, accessed on May, 25, 2015
3. Die Lage der IT-Sicherheit in Deutschland 2014, Bundesamt für Sicherheit in der Informationstechnik, 2015
4. BBC: Rail signal upgrade 'could be hacked to cause crashes', April, 24, 2015, http://www.bbc.com/news/technology-32402481, last accessed on May 20th, 2015
5. EN 50159 Railway applications, Communication, signaling and processing systems –Safety related communication in transmission systems, September 2010
6. EN 50129 Railway applications, Communication, signaling and processing systems – Safety-related electronic systems for signaling, February 2003
7. ISO/IEC 15408 Information technology — Security techniques — Evaluation criteria for IT security, 2009
8. IEC 62443: Industrial communication networks - IT security for networks and systems, series of 12 standards (planned), see http://en.wikipedia.org/wiki/Cyber_security_standards
9. Commission Regulation (EC) No 402/2013 of 30 April 2013 on the common safety method for risk evaluation and assessment and repealing Regulation (EC) No. 352/2009, http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32013R0402&from=DE, last accessed on April, 14th, 2014
10. DIN V VDE V 0831-104: Electric signaling systems for railways – Part 104: IT Security Guideline based on IEC 62443 (in German), 2015
11. Braband, J., Bock, H., Milius, B. und Schäbe, H.: Towards an IT Security Protection Profile for Safety-related Communication in Railway Automation, in: Ortmeier, F., Daniel, P. (eds.) Computer Safety, Reliability and Security, Proc. SAFECOMP2012, Springer, LNCS 7612, 2012, 137-148
12. Braband, J., Schäbe, H.: Probability and Security – Pitfalls and Chances (mit Schäbe, H.): in Proc. Advances in Risk and Reliability Technology Symposium 2015, Loughborough, 2015

7 Appendix: Relationship between SL 1 and Functional Safety

This annex reports in detail the result of a comparison between the SL 1 requirements (Security Requirements (SR) in compliance with IEC 62443-3-3 and those of EN 50129, EN 50128 and EN 50159. As SL 1 is only intended to offer protection against unintentional or random attacks, it may be presumed that safety-related systems that have to offer protection against foreseeable misuse and operating errors already fulfil a large proportion of the requirements. For example, EN 50129 also already protects against random errors and unintentional disruptions, similar to EN 50159 for Category 1 and Category 2 against corresponding transmission errors. Further measures are only necessary where unauthorised access cannot be ruled out (Category 3).

At the same time, the difference between SL 1 and SL 2 should be pointed out. Additional requirements that are added for SL2 are printed in italics.

This comparison does not mean that all requirements for SL 1 are already covered in EN 50129. At least the requirements that are normally not fulfilled by safety systems should be adopted as requirements in future. Table 1 gives a list of the requirements which are either covered by the safety standards or are not relevant from a safety point of view.

Ref.	Title	Requirement for SL 1	Fulfilment by safety standards
SR 1.1	Identification and authentication of persons	The automation system must have the ability to identify and authenticate all human users (persons). The automation system with its corresponding capabilities must assert the identification and authentication at all interfaces of persons who want to achieve access to the automation system, that would allow them to separate obligations and restrictive assignment of authorisations, in accordance with the applicable IT security guidelines and processes. *RE1 The automation system must have the ability to uniquely identify and authenticate all human users.*	This is required by EN 50129 B.4.6. This standard in particular requires that protective measures have to be taken with regard to – oversights by authorised personnel, and – intentional changes by unauthorised personnel. NB The requirements need not necessarily be technically implemented, but can also be fulfilled by corresponding organisational measures.
SR 1.2	*Identification and authentication of software processes and devices*	*The automation system must have the ability to identify and authenticate all software processes and devices and function units. The automation system with its corresponding capabilities must assert the identification and authentication at all interfaces of software processes and devices that want to achieve access to the automation system, that would allow them to restrictively assign authorisations, in accordance with the applicable IT security guidelines and processes.*	Generally, access protection is understood in a wide sense in EN 50129, just like in EN 50159.
SR 1.3	User account management	The automation system must have the ability to manage and administer all user accounts of authorised users, which includes opening new accounts and activating, modifying, blocking and deleting accounts.	This is not required explicitly, but is an implicit conclusion from EN 50129 B.4.6.
SR 1.4	Identifier management	The automation system must have the ability to manage identifiers and IDs of different kinds and conditions according to users, groups, roles or different interface types of the automation system.	This is not required explicitly, but is an implicit conclusion from EN 50129 B.4.6.
SR 1.5	Authenticator management	The automation system must possess the following capabilities and must implement them: a) initialising the authenticator's content, i.e. the means of confirming a user's identity; b) changing all default authenticators after installation of the automation system; c) changing or renewing all authenticators; and d) protecting all authenticators from an unauthorised disclosure and modification during storage or transmission.	This is not required explicitly, but is an implicit conclusion from EN 50129 B.4.6.
SR 1.6	Management of wireless access processes	The automation system must have the ability to identify and authenticate all users communicating by wireless means (persons, software processes and devices). *RE 1 The automation system must have the ability to uniquely identify and authenticate all users communicating by wireless means (persons, software processes and devices).*	For SL1, this requirement is contrary to EN 50159 B.1. A radio transmission system would generally be assigned to Category 3 and would need cryptographic protection, i.e. more than SL1. Therefore, this requirement is not relevant for SL1.
SR 1.7	Security of authentication by passwords	Automation systems that use passwords for authentication must have the ability to enable configuration of password security by means of password length (with a given minimum length) and diversity of characters.	This is not required explicitly, but is an implicit conclusion from EN 50129 B.4.6.
SR1.8	*PKI certificates*	*If a public key infrastructure (PKI) is used, the automation system must have the ability to operate such a public key infrastructure in accordance with common conventions or it must be able to obtain public key certificates from an existing public key infrastructure.*	Covered in EN 50159 for Category 3 if asymmetrical methods are used.
SR 1.9	*Security of asymmetrical crypto systems*	*In automation systems that use asymmetrical crypto systems (public key crypto systems) for authentication, the automation system must have the following capabilities:* *a) The ability to validate certificates by checking the validity of a given certificate's signature* *b) The ability to validate certificates by building up a certification path to a recognised certification agency (CA) or, in the case of a self-signed certificate, by issuing mutual confirmations to all hosts that communicate with the key holder for whom the certificate was issued* *c) Checking certificates to determine whether they are on a certificate blacklist (or a revocation list)* *d) Bringing about secure storage and monitoring of the associated private key by the user (person, software process or device)* *e) The ability to map the authenticated identity to a user (person, software process or device)*	Addresses in EN 50159 for Category 3 if asymmetrical methods are used, but not in relation to all details.
SR 1.10	Feedback from the authenticator	The automation system must have the ability to suppress (blacken) feedback messages generated by the authenticator during the authentication process.	This is not explicitly required, but is easy to realise.

SR 1.11	Failed login attempts	The automation system must have the ability to assert a limit to the number of failed successive login attempts (person, software process or device) within a configurable time. The automation system must have the ability to block physical or logical access for a specified time or it must allow an administrator to lift this block again after this time had been exceeded. For system accounts on whose behalf critical services or servers are operated, the automation system must provide for the ability to forbid interactive logins.	This is not explicitly required, but is easy to realise.
SR 1.12	Reference to system use	The authentication system must have the ability to refer, even before authentication, to the rights and obligations linked with use of the system. A 'System Use Notification' is displayed for this purpose. It must be possible for authorised personnel to configure this display.	This is not explicitly required, but is easy to realise if wished by the operator.
SR 1.13	Access through un-trustworthy networks	The automation system must have the ability to monitor and control all kinds of access to the automation system through untrustworthy networks. *RE1 The automation system must have the ability to deny access through unreliable networks, but the desire is approved by an instance authorised to do this.*	In SL1, only Category 1 and 2 networks come into consideration. In 1 there are only known users, i.e. the requirement is unnecessary. In Category 2, although the users are not all known, they are trustworthy.
SR 2.1	Enforcing authorisation	At all interfaces, the automation system must ensure enforcement of the authorisations assigned to all human users; as a result, these persons become authorised and are enabled to control the automation system in such a way that separation of duties and restrictive authorisation assignment can also be asserted. *RE 1 At all interfaces, the automation system must ensure enforcement of the authorisations assigned to all users (persons, software processes or devices); as a result, these users become authorised and are enabled to control the automation system in such a way that separation of duties and restrictive authorisation assignment can also be asserted.* *RE 2 The automation system must possess the ability and must make it possible for an authorised user or an authorised role to define and modify mapping of the authorisations of all human users to roles.*	This is derived from EN 50129 B.4.6.
SR 2.2	Use control and monitoring in the case of radio connections	In the case of radio (wireless) connections into the automation system, the automation system must possess the ability to authorise, monitor and also fulfil restricted use in accordance with the security conventions that are generally common in industrial practice.	This requirement is contrary to EN 50159 B.1. A radio transmission system would generally be assigned to Category 3 and would need cryptographic protection, i.e. more than SL1.
SR 2.3	Use control and monitoring in the case of portable and mobile devices	The automation system must possess the ability to automatically implement and execute configurable restrictions of use: a) Preventing use of portable and mobile devices b) Demanding a context-specific authorisation c) Restricting transmission of data and code from and to portable and mobile devices	In EN 50129, mobile devices are treated just like other devices if they perform fail-safe tasks. The specific requirements must be derived from a hazard analysis, however.
SR 2.4	Mobile code	If mobile code techniques are used, the automation system must be able to assert restricted use, taking into account the damage that can possibly be caused in the automation system. These abilities include: a) Preventing execution of mobile code b) Demanding clean authentication and authorisation of the code source c) Limiting transfer of mobile code to and from the automation system d) Monitoring the use of mobile code	Mobile code is not allowed in safety-related systems because it is not covered by validation and approval.
SR 2.5	Session blocking	The automation system must possess the ability to prevent further access to the system by blocking the session after an adjustable inactivity time or by manual intervention. The session must remain blocked until the session owner or another authorised person restores access by again initiating the identification and authentication process intended for this purpose.	This is not explicitly required, but is easy to realise.
SR 2.6	*Ending a remote session*	*The automation system must possess the ability to end a remote session either automatically after an adjustable period of inactivity or it must make it possible for the session to be ended manually by the user who initiated it.*	This is not explicitly required, but is easy to realise.
SR 2.8	Verifiable events and their recording	The automation system must possess the ability to generate audit data (data recorded during computer and network monitoring, information technology measurements) concerning the IT security achieved in the following categories and to record such data as audit records: access control, flawed queries, incidents in the operating system, incidents in the automation system, incidents during backup and recovery of data, potential reconnaissance and incidents during audit report creation. The individual audit reports must contain the following information: Time of the incident, incident source (designation of the device, equipment, software process or user account in which the incident is taking place or has taken), category, type, incident number and result.	Although data logging is a common practice, it is not required normatively because such data is generally not considered to be relevant to safety.

SR 2.9	Storage capacity for audit records	The automation system must provide adequate storage capacity for storing audit records in accordance with generally recognised recommendations for log management (archiving of incident logs) and system configuration. The automation system must ensure that not too much storage capacity is maintained.	See above.
SR 2.10	Response to failed audit data processing	If it should transpire that the audit data (data recorded during computer and network monitoring, measured results regarding information technology processes in IACS) is no longer processed at all or no longer correctly, the automation system must possess the ability to inform operating personnel of this and it must prevent the loss of essential services and functions. As a response to failed processing of audit data, the automation system must possess the ability to initiate suitable remedies in accordance with generally recognised industrial conventions and to support them.	See above.
SR 2.11	*Time stamp*	*The automation system must assign a time stamp to the audit records generated.*	As detailed in EN 50159, time stamps can be used, but are not required.
SR 3.1	Communication integrity	The automation system must possess the ability to preserve the integrity of information transferred.	Protecting the integrity of the message stream is a basic requirement of EN 50159.
SR 3.2	Protection against harmful code	The automation system must possess the ability to take precautions against harmful code or unauthorised software; corresponding mechanisms should detect and report such harmful code and should defuse any negative impacts. These protective mechanisms must be updated. *RE 1 The automation system must possess the ability to use processes for protection against harmful code at all entry and exit points.*	In SL1, IEC 62443 assumes untargeted attacks, the viruses, etc. are not specifically directed at the system. EN 50128 15.4.6 requires protection of software against unintentional or random modification and this suffices for SL1.
SR 3.3	Verification of IT security functionality	The automation system must possess the ability to verify the intended operation of the IT security functions and must report whenever anomalies are detected during factory acceptance testing (FAT), during site acceptance testing (SAT) and during a scheduled maintenance operation. These security functions must comprise all functions that are needed to fulfil the information technology security requirements defined in this standard.	In the case of safety systems, this requirement is covered by validation in compliance with EN 50128.
SR 3.4	Software and information integrity	The automation system must possess the ability to detect, record, report and protect against unauthorised changes to software and stored inactive or archived data.	EN 50128 13 requires protection of software against unintentional or random modification.
SR 3.5	Input validation	The automation system must validate the syntax and the contents of indirect inputs into an industrial process control system and of direct inputs with direct impacts on the automation system.	Plausibility checks are required by EN 50129 E.5.1 and also by the principle of defensive programming in EN 50128.
SR 3.6	Deterministic output	The automation system must possess the ability set outputs to a predetermined status if no normal operation can be maintained any more as a result of an attack.	In accordance with EN 50129 B3.4, a safe status must be assumed in the event of a fault, including avoidance of unsafe outputs.
SR 3.7	Error handling	The automation system must detect errors and must handle error states in such a way that an effective remedy is possible. At the same time, steps must be taken to ensure that no information is disclosed that can be used by enemies to attack the IACS unless the disclosure of this information is indispensable to remedy the problems in good time.	EN 50129 and EN 50159 do not contain any specific requirements in this respect.
SR 3.8	*Session integrity*	*The automation system must possess the ability to preserve the integrity of sessions. The automation system must reject use of invalid session identifiers (IDs).*	This is required in EN 50159.
SR 3.9	*Protection of audit information*	*The automation system must protect verified and recorded incidents (audit information) and audit tools (insofar as available) against unauthorised access, modification and deletion.*	EN 50129 and EN 50159 do not contain any specific requirements in this respect.
SR 4.1	Confidentiality of information	The automation system must possess the ability to preserve the confidentiality of information for which a read authorisation is expressly required, be it in transit or in the idle state. *RE 1 The automation system must possess the ability to preserve the confidentiality of information or data that is in the idle state and protect data that is routed through an untrustworthy network during a remote session.*	Confidentiality is not normally required for railway applications. Processes that are not safety-related may also access information.
SR 4.2	*Information constancy*	*The automation system must possess the ability to permanently delete all information on data media for which a read authorisation was expressly required and which are to be taken out of operation or shut down.*	EN 50129 and EN 50159 do not contain any specific requirements in this respect.
SR 4.3	Using encryption	If encryption is required, the automation system must use cryptographic algorithms for the size, the mechanisms of key creation and management of keys in accordance with the security conventions and recommendations generally recognised in information technology.	Generally not required in the case of Category 1 or 2 in accordance with EN 50159 Annex C.

SR 5.1	Network segmentation	The automation system must possess the ability to logically separate automation systems from non-automation systems and to logically separate critical automation systems from other automation systems. *RE 1*: The automation system must possess the ability to physically separate automation systems from non-automation systems and to physically separate critical automation systems from other automation systems.	This is a basic requirement of EN 50159 7.3.7.2 and is generally warranted by the safety protocol.
SR 5.2	Protection of the zone boundary	The automation system must possess the ability to monitor communications at zone boundaries and to intervene, if necessary, to be able to execute the departments defined in the risk-based zone and conduit model. *RE 1 The automation system must possess the ability to always reject network traffic and to permit it only in exceptional cases.*	Only Category 1 and Category 2 networks may be used for SL1. These form a single zone with uniform IT security requirements and so no splitting is necessary and this requirement does not make sense for SL1.
SR 5.3	Restriction of general communication between persons	The automation system must possess the ability to prevent exchange of messages between persons that are sent by users or systems outside the control system and are received by persons inside the control system.	Generally, voice communication is not part of the safety system. However, this requirement should be exported to the operator.
SR 5.4	Partitioning applications	The automation system must possess the ability to partition data, applications and services depending on the complexity of the zone model to be realised.	This is a requirement of EN 50129 E.2.1.
SR 6.1	*Access to audit logs*	The automation system must possess the ability to grant read access to stored audit logs to authorised persons and tools.	Although data logging is a common practice, it is not required normatively because such data is generally not considered to be relevant to safety.
SR 6.2	*Continuous monitoring*	*The automation system must possess the ability to continuously monitor the performance and behaviour of all IT security mechanisms and, to this end, to use the security conventions and recommendations that are generally recognised in information technology, thus being able to detect and report on any security violations early on.*	This is explicitly required in EN 50159 if the IT security functionality has not been developed in accordance with EN 50129, i.e. in particular in the case of commercial components.
SR 7.1	Protection against DoS attacks	The automation system must possess the ability to continue working in a restricted mode of operation during a DoS attack. *RE* 1 The automation system must possess the ability to control the traffic load (for example by limiting the data transfer rate) so that the impact of a provoked inundation with data leading to triggering of a reduced availability can be mitigated.	This requirement is normally not contained in safety standards. In railway, however, there is normally a fallback level after failure of technology. In future, IT security aspects may have to be considered in the design of the fallback level.
SR 7.2	Resource management	The automation system must possess the ability to counteract exhaustion of resources; to this end, security functions would possibly have to be granted fewer resources.	This requirement is normally not contained in safety standards.
SR 7.3	Backups of the automation system	The automation system must be capable of storing and archiving backup copies (backup) of critical files and data from the user and the system levels (including information about system status) in a secure location without detrimentally influencing ongoing operation of the system. *RE* 1 The automation system must possess the ability to check the operability (reliability) of backup mechanisms.	This requirement is normally not contained in safety standards.
SR 7.4	Restart and recovery of the automation system	The automation system must possess the ability to restart after an interruption or a failure and to return to a known secure state.	This is required in EN 50129 B5.2.
SR 7.5	Emergency power supply	The automation system must possess the ability to switch to an emergency power source, or to return to a normal supply source from it, without exerting any detrimental impact on the existing security state or a documented restricted mode of the IACS.	This requirement is normally not contained in safety standards.
SR 7.6	Network and security settings	The automation system must possess the ability to be configured as provided for in the instructions included by the supplier of the automation system; this applies in particular to recommended network and security settings. The automation system must provide an interface to the current network and security settings.	EN 50129 or EN 50128 requires configuration management as part of quality management. The requirements for configuration are part of the safety application conditions.
SR 7.7	Restrictive functionality assignment	The automation system must possess the ability to specifically suppress the application and use of unnecessary functions, ports, protocols or services or at least to restrict these applications.	EN 50128 requires complete tests. Railway software may only contain (activated) functions that are required in accordance with the specification. EN 50128, Section 7.3.4.7 can be referenced with regard to pre-existing software. Nevertheless, the result in certain circumstances for COTS components such as a switch is that certain functions are deactivated.
SR 7.8	*List of the automation system's components*	*The automation system must possess the ability to issue a list of all currently installed components of the automation system with the relevant characteristics and features.*	This requirement is normally not contained in safety standards.

Applying MILS principles to design connected embedded devices supporting the cloud, multi-tenancy and App Stores

Embedded, connected devices are nothing new. Developers have been establishing Machine to Machine communications for 20 or 30 years, long before the Internet of Things was ever conceived.

But until quite recently, embedded applications have tended to be static, fixed function, device specific implementations. In the current environment of ever quickening technological change, morphism and evolution are the order of the day. Now we see manufacturers and service providers all seeking to monitor, upgrade, enhance and supplement software implementation on a continuous basis.

In the enterprise world, cloud computing and “X”-as-a-service (PaaS, IaaS, SaaS) architectures have dramatically changed the economics of IT such that the infrastructure is now a utility or service rather than a capital investment. This has enabled significant innovations in service, including App Stores - libraries of products and services that are supported by the infrastructure and provide single instances of cloud-based software for the benefit of multiple “tenants”.

In the IoT, embedded devices are somehow connected to the “cloud”. However, the benefits of the IoT are unlikely to be realized unless the connected embedded devices are capable of supporting the multi-tenanted architectures that are common in enterprise IT. This implies that IoT devices need to provide secure separation between the different tenants accessing the IoT infrastructure.

For example, imagine a car as an IoT gateway device, with many demands on its systems infrastructure. Of course, the manufacturer will be first in the queue, providing software updates to security critical aspects of the vehicle and monitoring its condition. But alongside this core functionality there will also be a host of Apps from the Store to enable and entertain. The navigation service provider, accessing continuously evolving road data. Insurance applications, monitoring probationary drivers to help minimize premiums. Media streamers with the latest movies. Games, advertisements, and a host of other possibilities as yet unrealized.

Multi-tenancy without security is not enough. As if to highlight that point, at the 2010 IEEE Symposium of Security and Privacy, researchers from the University of Washington and the University of California – San Diego remotely accessed the systems on a late model car to unlock the doors, start and stop the engine, lock brakes and disable windscreen wipers – all potentially life threatening actions[1].

In their paper on the subject, the researchers observed that

“While the automotive industry has always considered safety a critical engineering concern (indeed, much of this new software has been introduced specifically to increase safety, e.g., Anti-lock Brake Systems) it is not clear whether vehicle manufacturers have anticipated in their designs the possibility of an adversary. Indeed, it seems likely that this increasing degree of computerized control also brings with it a corresponding array of potential threats. Compounding this issue, the attack surface for modern automobiles is growing swiftly as more sophisticated services and communications features are incorporated into vehicles.”

It is precisely these conflicting demands for security and accessibility which makes this and similar environments quite so challenging.

[1] Experimental Security Analysis of a Modern Automobile http://www.autosec.org/pubs/cars-oakland2010.pdf

Designing an IoT Gateway

Generalizing the automotive example to the wider Internet of Things, it is useful to consider exactly what properties are desirable in an IoT Gateway. Traditional embedded designs using COTS embedded operating systems can present many compromises.

Traditional embedded gateway designs rely on monolithic architectures, such that applications are hosted by a single operating system and all I/O support, management controls, and security controls are integrated into the operating system kernel. This monolithic construction creates the following challenges:

- **Fragile Single Point of Failure**
  IoT gateways are highly susceptible to attacks and failure due to their wide exposure to the internet across so many interfaces supporting a great deal of complex functionality. Because monolithic designs host all applications, perform I/O, and management functions, any failure in security policy or kernel coding flaw can jeopardize the security and availability of the system which will greatly limit the ability to operate in stringent safety markets.
- **Weak Separation**
  In order to provide a platform as a service a strong degree of separation will be needed to prevent collocated applications from compromising the privacy or availability of other applications. Relying on a monolithic operating system to provide separating computing environments is risky given the number of ways monolithic operating systems can fail or be subverted.
- **Limited I/O Support**
  With a monolithic design, all sensor and network interfaces drivers and I/O stack support must be built into the operating system kernel. If drivers do not exist for the selected OS, it can be difficult to support all varieties of desired sensor support and network interfaces.
- **Limited Application Support**
  Certain applications may only be available for certain operating systems. Choosing an OS that provides the best IO support, application support, ideal deterministic behaviour, and secure design may not be feasible.
- **Limited Online Maintenance**
  Due to highly complicated interdependencies of functionality in monolithic operating systems, the ability to patch or upgrade kernel functionality while maintaining platform operation is highly limited. The majority of kernel maintenance procedures require platform reboots which pose significant challenges for platforms that strive for full autonomy.

  The majority of issues in relying on a monolithic operating system as the foundation of an IoT gateway stem from limited application and IO support, and exposure to a single point of failure. A better approach is to have a more modular design that can provide more interoperability options and achieve higher levels of reliability and assurance.

Given these traditional designs are compromised, it is useful to consider relevant academic principles as a basis for a better solution.

The application of MILS principles

The solution to this conundrum lies in the MILS (Multiple Independent Levels of Security/Safety) initiative. MILS is a high-assurance security architecture based on the concepts of separation and controlled information flow; implemented by separation mechanisms that support both untrusted and trustworthy components; ensuring that the total security solution is non-bypassable, evaluatable, always invoked and tamperproof[2]. A MILS compliant distributed secure system for the connected car will therefore comprise of high-assurance components and applications which can be independently

[2] https://en.wikipedia.org/wiki/Multiple_Independent_Levels_of_Security

developed, modularly combined, evaluated and certified to adhere to these principles, in all aspects of its development.

This paper outlines the principles and application techniques surrounding three enabling technologies.

1. Least Privilege Separation Kernel Hypervisor,
2. MILS based network components for network traffic encryption and the consolidation of multiple encryption tunnels
3. MILS based data handling components to encrypt data and consolidate multiple storage partitions

Least Privilege Separation Kernel

The foundation of any such MILS compliant system is the Separation Kernel, a concept first mooted in 1981 by John Rushby[3]. A separation kernel consists of "a combination of hardware and software that permits multiple functions to be realized on a common set of physical resources without unwanted interference". By implication that suggests that the only interaction between the "security blocks" is by design, and that primary information flow will be from high to low security blocks (Figure 1).

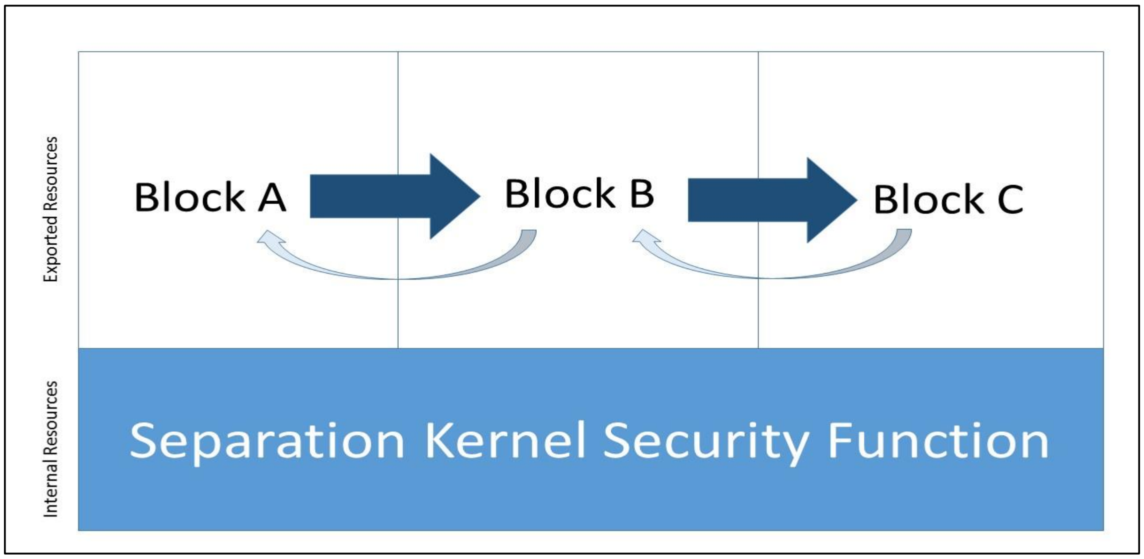


Figure 1. Primary information flow is from high to low security blocks – but some flow is required in the opposite direction.

A few years earlier in 1975, Saltzer and Schroeder[4] established a similar set of principles based on the idea of modularization, noting that "Every program and every user of the [operating] system should operate using the least set of privileges necessary to complete the job". The concept of a separation kernel clearly goes some way towards that goal, but because separation kernels are traditionally based on resource isolation there was insufficient granularity to take that principle to its conclusion.

[3] Design and Verification of Secure Systems. John Rushby, Computer Science Laboratory, SRI International. 1981.

[4] Saltzer and Schroeder, The Protection of Information in Computer Systems, ACM Symposium on Operating System Principles (October 1973)

That coarse granularity, coupled with an inevitable requirement for at least some flow of information from low to high security blocks, results in considerable emphasis being placed on developers if the boundaries are not to be breached.

The idea of marrying the two was therefore proposed by Levin, Irvine and Nguyen[5] resulting in the concept of a Least Privilege Separation Kernel, where both resources and subjects (executable entities) could be modularized. In this way, no subject needs to be given more access than required to allow the desired flows (Figure 2).

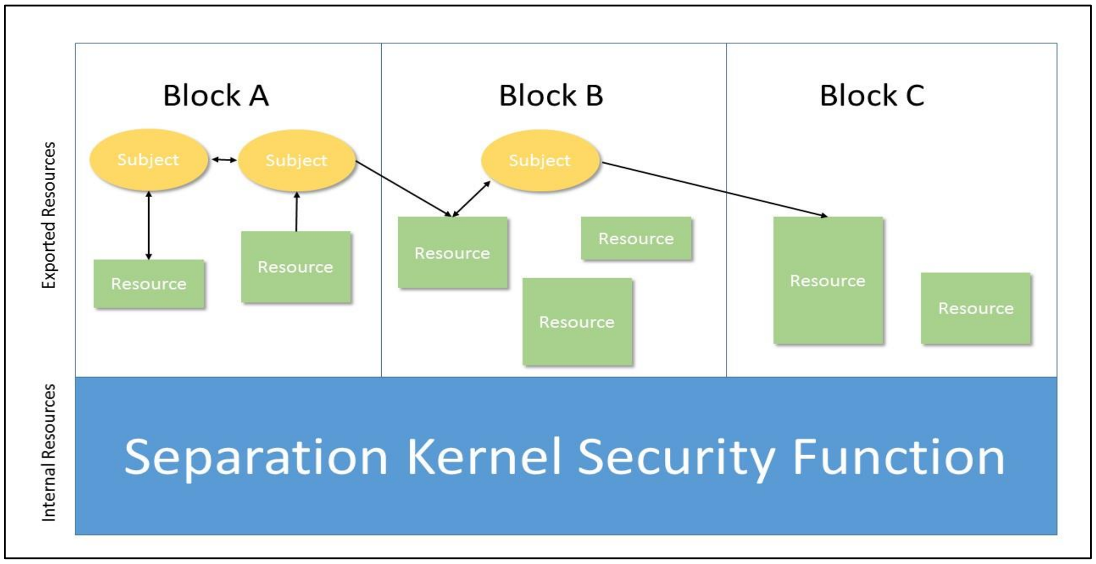


Figure 2. Superimposing Least Privilege principles on to those of a Separation Kernel combines per-subject and per-resource flow control granularity

In practical terms, the modularity of this approach enables the creation of de-privileged configuration utilities, device drivers, guest virtualization, device virtualization and management services. Hypervisor functionality is almost a by-product of this architecture, in that the subjects illustrated in Figure 2 can consist of anything from a minimally sized, “bare metal” application to a fully featured Real Time or General Purpose Operating System.

MILS based network and data storage components

In the context of our automotive example, this Least Privilege Separation Kernel Hypervisor may well be a well-founded principle, but it is only useful if it can be applied to provide the secure environment we need to ensure that the car as an “IoT gateway device” is not vulnerable to the attacks of the troublesome researchers and others with more malicious intent.

This has to be achieved, however, in the context of an existing infrastructure consisting very largely of the public internet. The onus for security therefore falls on the gateway devices – and in our case, that means the systems contained within the car.

There are also cost considerations to consider, particularly in the case of the ever expanding functionality of the modern car. To that end, it is important that the high build and maintenance costs associated with multiple vehicle networks should be avoided.

[5] T. E. Levin, C. E. Irvine, and T. D. Nguyen. Least privilege in separation kernels. In J. Filipe and M. S. Obaidat, editors, *E-business and Telecommunication Networks; Third International Conference, ICETE 2006*

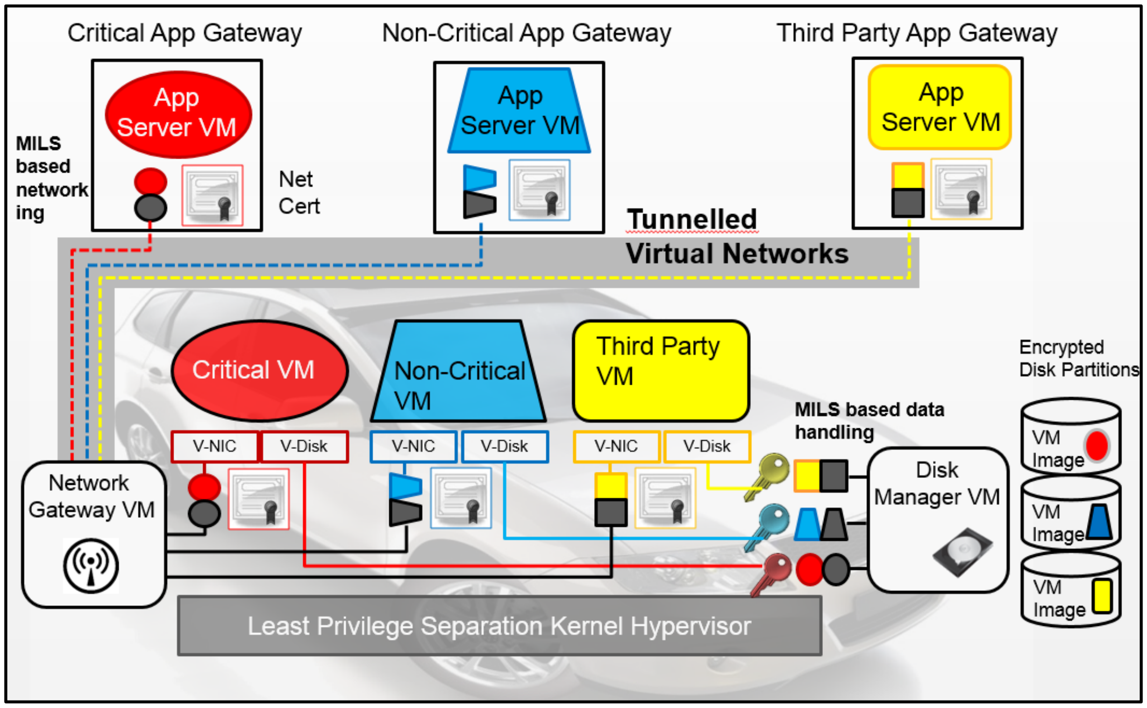


Figure 3. Superimposing Least Privileged subjects to handle data storage and networking functionality on the Least Privilege Separation Kernel Hypervisor.

Figure 3 shows how such a system could be designed in practice. The networking and data handling are implemented as minimalist least privilege “subjects”, tightly coded and running as “bare metal” applications to minimize their footprint and hence vulnerability.

The configuration of the system is completed statically at installation time, and the facilities for modifying that configuration do not exist once the system is running.

Three virtual machine (VM) subjects provide the means to run the Apps, possibly mapped to the individual cores of a multicore processor where there is hard real time dependency.

Encryption keys ensure the integrity of three tiers of data security, while the Least Privilege Separation Kernel provides the underpinnings to guarantee that the trusted code base underpinning the system is minimalized.

The net result is a robust solution, providing resilient application interfaces to prevent malicious software from subverting the virtual software architecture. It ensures the integrity of critical applications by protecting them from possible corruption from other application partitions. Costs are minimized and yet confidentiality is guaranteed through the application of a single network structure. And it can be ensured that vehicle applications are genuine, and that network encryption cannot be bypassed by server vehicle application VMs.

In summary, by applying the MILS principles throughout, the security/accessibility conundrum is resolved.

The Gateway based on the Least Privilege Separation Kernel Hypervisor

It is useful to summarize the pertinent capabilities and properties offered by Least Privilege Separation Kernel Hypervisors (more succinctly, “Separation Kernel Hypervisors”) in the context of the design of IoT gateways in general, and by implication, the connected car in particular.

- **Virtualization**
  Separation Kernel Hypervisors offer the ability to run multiple variants of guest operating systems concurrently on the same platform. This capability provides many options for hosting applications from multiple different operating systems and supporting a variety of different sensor devices and network interfaces. For example, through virtualization a design can be made where timing critical applications such as load-balancing and failover protocols run in real-time operating systems while running lessor critical applications in more feature rich operating systems. Virtualization is a key capability in dealing with interoperability challenges. Virtualization also plays a key role in supporting edge computing and platforms as a service, allowing analysis tools and tenants to use any operating systems and application best suited for the target task.
- **Bare-metal Applications**
  Some Separation Kernel Hypervisors provide the ability to run applications without the assistance of an operating system. Bare-metal applications are helpful for designs that require high performance or high assurance. Removing the dependency of an operating system reduces application overhead. For deployments that face stringent safety and security requirements, code reviews of critical applications may be necessary. Bare-metal applications can greatly aid the code analysis process by removing the complex code dependencies on the operating system.
- **Full Mediation**
  Separation Kernel Hypervisors provides the ability to explicitly monitor and control all platform transactions. This allows system architects to gain full awareness of the state of the system that can be used to help improve platform availability and assurance.
- **Isolation**
  Separation Kernel Hypervisors partition hardware resources and assign them to either Guest OSs or bare-metal applications according to a specification defined by a system architect. Separation Kernel Hypervisors provide the strongest degree of protection of resources using the native capabilities of the CPU. Strong resource isolation is a key enabler for achieving high availability.
- **Robustness**
  Because Separation Kernel Hypervisors serve as the central host of the computing platform, they are designed specifically to be highly reliable and resilient to malicious techniques of subversion. Separation Kernel Hypervisors achieve this property by relying on simple least privilege internal designs with limited functionality and limited exposure to hosted software. Separation Kernel Hypervisors tend to be ten to hundreds of times smaller than monolithic operating systems.
- **Determinism**
  Separation Kernel Hypervisors provide the ability to control explicit execution schedules for guest operating systems and bare-metal applications. Coupled with the ability to isolate resources for software modules, Separation Kernel Hypervisors can guarantee the availability of a system, ensuring any critical application can never be pre-empted or starved by competing applications. The deterministic control Separation Kernel Hypervisors have over a computing platform is a key enabler for designing highly reliable systems.
- **Certification**
  Separation Kernel Hypervisors are designed to support environments that face both security and safety regulation. Relying on the intrinsic properties and core capabilities of a Separation Kernel Hypervisor system architects are given the tools needed to design and implement

solutions greatly exceed the levels of reliability compared to a solution running on a monolithic operating system.

Least Privilege Separation Kernel Hypervisor under attack

All of these features clearly have merit, but ultimately the Separation Kernel Hypervisor itself is the critical component of the architecture; the trusted code base that makes the rest of the architecture possible. That make it a primary target for attack.

It would be bold or foolish to claim that it could never be compromised, but the theoretical principles which guided its design also highlight why that threat is optimally defended.

All attacks require an attack vector; a means of entry, commonly a NIC or device driver. LynxSecure offers no such direct attack vector because in accordance with the principles of a least privilege separation kernel, device drivers and communication mechanisms such as NICs are all placed within VMs.

The possible configuration permutations of LynxSecure are almost countless. The superimposition of Least Privilege principles on Separation Kernel blocks results in fine per-subject and per-resource flow-control granularity, and so communication paths only exist if the system architect wants them.

Where they must exist, should there be a concern that communication between VMs by means of a virtual Ethernet and its well-known protocol, then the architect might prefer to take advantages of the proprietary “hypercalls” API which adds an element of “security by obscurity” to the design.

Figure 3 shows an example of a communications mechanism between VMs by means of virtual network cards. That clearly involves communicating through LynxSecure, but not with it. In other words, for this to represent a vulnerability there would have to be coding deficiencies to compromise the integrity – and the very fact that it is so small and has been certified to very high standards means that the risk is tiny.

Finally, LynxSecure is statically configured. The mechanism required to modify that configuration does not exist at run time, and so there is no possibility of the installed configuration being modified by an attacker.

In summary; there has to be a communications path through the gateway for it to fulfil its purpose, and that implies the existence of possible means of compromise. But here, the risks are kept to an absolute minimum in accordance with the principles on which its design is founded.

Least Privilege Separation Kernel Hypervisor in practice

The example of an automotive application is highly topical (figure 4).

ANDY GREENBERG SECURITY 07.21.15 6:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Applying MILS principles to design connected embedded devices supporting the cloud, multi-tenancy and App Stores

Figure 4. The vulnerability of the connected car is a highly topical subject, and a thorny one for the automotive industry

It is perhaps no surprise that it is presently under review by a number of vehicle manufacturers and their tier one suppliers, and that at the time of writing the commercial relationships involved mean that detailing their practical application here is not possible.

The security issue highlighted is new to the automotive sector, because until recently cars have been secure simply by virtue of being isolated. However, that is not true of all other fields of endeavour, and there are practical applications of this technology dating back for almost a decade. This state of affairs therefore has the potential to offer the automotive and other sectors a proven and seasoned solution to a new and challenging problem.

One field where security has always been paramount in in the military, and it is perhaps natural that Separation Kernel Hypervisors have therefore found many of their earliest practical applications in that field. Such applications clearly demand certification in accordance with military security standards. For example, a US armed forces program has used a Separation Kernel Hypervisor as the security foundation for an embedded mission critical security solution. This solution successfully underwent a security certification involving extensive penetration and other testing for use in DOD High Threat Environments, satisfying the following security requirements:

- Director of Central Intelligence Directive (DCID) 6/3 Protection Level (PL) 4 – separation of Secret to Secret/compartmented security domains in the presence of untrusted users.

- 8500.2 Mission Assurance Category I (MAC I) for systems handling information that is determined to be vital to the operational readiness or mission effectiveness of deployed and contingency forces in terms of both information system content integrity and timeliness/availability.

Although further details about this certification and others like it are classified, the artefacts created to achieve that certification are mostly generic and hence relevant to other certification processes.

Other “real life” applications include

- A UAV ground controller, providing a user interface and control platform for controlling unmanned vehicles. The use of a Separation Kernel Hypervisor in this environment provided
    - A path to certification,
    - Safety critical partitioning of the user interface from the control function,
    - Deterministic control, and
    - Flexible application options.
- An electronic “flight bag”, consisting of a cockpit user interface for features including a map display an electronic forms. In this case, the Separation Kernel Hypervisor provided
    - The separation of a low integrity user interface from a high integrity aircraft bus
    - The optimal OS for each application, permitting the use of state-of-the-art graphics
    - A certifiable approach to isolating fully virtualised OS

Separation Kernel Hypervisor Based Design Strategies – an example

Suppose that there is a requirement for an IoT gateway in a traffic control application. In its simplest form, the gateway in the system is providing local control for a set of traffic lights. Road sensors provide information about what is happening to the traffic flow, the gateway provides local control to action the immediate policies, and the policy decisions are derived from the command centre based on the information fed back to it from the gateway.

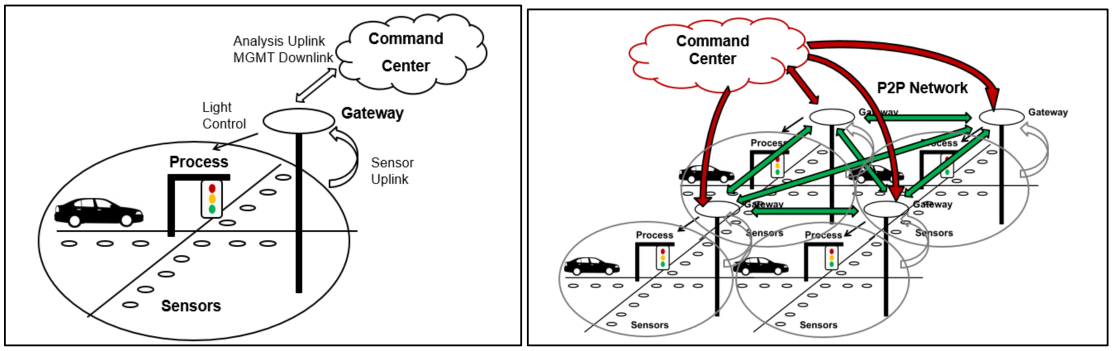


Figure 5. A simple IoT Gateway application providing local control for a set of traffic lights (left). More complex traffic control requires local peer-to-peer communication as well as policy decisions downloaded from a command centre (right)

As this system is expanded, so the intercommunication between gateways and the command centre becomes more complex. Immediate traffic flow issues are management by means of local P2P networking while the policy decisions are still downloaded from the command centre (figure 5)

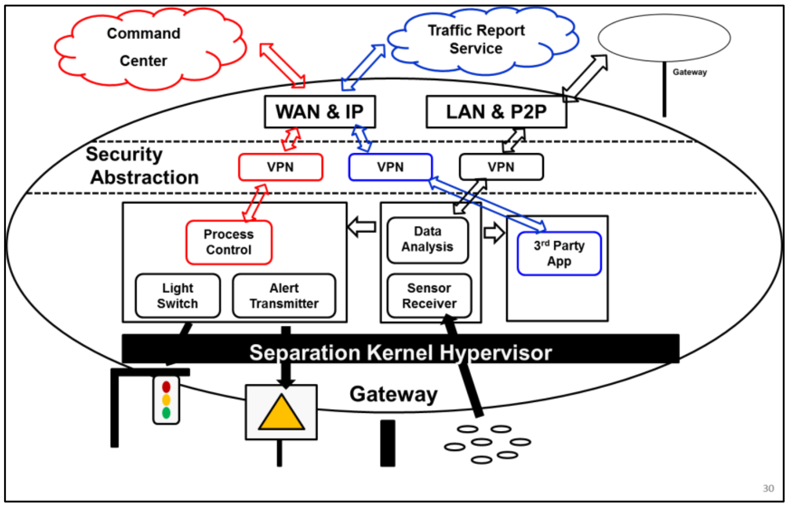


Figure 6. Ultimately the system requires multiple domains to permit the various tenants safe access to the same TCP/IP stack

Ultimately, the system requires third party access to collate data for traffic reporting; naturally a less critical domain requiring separation from the traffic control functionality of the system (figure 7).

In the context of this traffic control system, several attributes of the Separation Kernel Hypervisor based gateway are significant.

- Modular Composition

  Using the virtualization and bare-metal application capabilities, and relying on the isolation and deterministic properties of a Separation Kernel Hypervisor, platforms can be constructed out of a variety of software modules where individual functions of the platform can run in independent partitions and simple interfaces between modules can be defined so that the state of the system is well defined and understood.

Modular designs promote better options for interoperability and sustainability. For example, gateway maintenance routines such as new sensor device or network interface upgrades or bug patches can be performed without powering down the device or disrupting vital operational components.

In the traffic control system (figure 7), we see a partition hosting a VM dedicated to the local lights and signage, a second to the receipt and analysis of data, and a third transmitting data to a traffic report service. An additional VM running a bare metal application ("Security abstraction") takes care of the encrypted data from the different tenants.

- Heterogeneous Computing

  Relying on virtualization, architects can mix and match combinations of operating systems concurrently running on the same platform to host a wide variety of applications and I/O interfaces that may be exclusive to certain operating systems. Heterogeneous computing is an excellent strategy for supporting the most interoperable designs, and defending against proprietary vendor lock-in or obsolescence.

  The traffic control example might deploy a bare metal application for the Network Gateway VM and the traffic signal control, perhaps an RTOS for the data analysis, and a general propose OS for the traffic report data handling.

- Fault Tolerance

  Relying on the robust and highly available backbone of the Separation Kernel Hypervisor and modular design principles, system architects can create fault-tolerant designs using heart-beat and health inspection techniques. Some Separation Kernel Hypervisors allow guest operating systems to report their health status, policies can be set in a Separation Kernel Hypervisor to reboot or repair modules in the event of failed health status reports. Furthermore specialized custom modules can be created to inspect the health of other modules either through communication protocols or simply giving inspection modules access to view operating system and bare-metal application memory resources.

- Security Abstraction

  A good architectural approach is to design systems such that applications run in separated environments from security sensitive applications to prevent security controls from being bypassed or infiltrated. In the traffic control system, the "Security Abstraction" layer takes the form of a VM running a bare metal application takes care of the encrypted data from the different tenants to prevent users from disabling or bypassing the crypto function.

Conclusions

Monolithic architectures pose significant challenges in the ability to support stringent safety standards and meeting demanding market needs. With the use of a Separation Kernel Hypervisor and modular design techniques, IoT gateway vendors are given many options for building highly interoperable, reliable, secure, and sustainable solutions using low cost COTS components.

The automotive industry in particular has suffered by moving from a world where system isolation offers the ultimate security protection, to the connected car where that past assumption makes it especially vulnerable.

Such industries cannot discard all existing code and applications, and re-write them overnight. Separation Kernel Hypervisors offers the proven, certified technology to provide separation of that vulnerable software from the dangers of external public access.

This is not new technology. It is proven and has been certified and in use in the US military for almost a decade. It is not an operating system; not even a cut down operating system. It is a separation kernel, leveraging hardware virtualization to minimize both the trusted code base and the attack surface.

Session 17

Tool support

Thursday 28th, 15:00 – Auditorium St Exupery

Accelerate the Development of Certified Software for Train Control & Monitoring Systems

Franck CORBIER

Dassault Systèmes, 35 rue Haroun Tazieff, 54320 Maxéville, France

Keywords

Transportation, Railway, TCMS - Train Control & Monitoring System, Model based System Engineering, EN50128, SSIL – Software Safety Integrated Level, EN50128, PLCopen, ControlBuild, Automatic Code Generation, Software Certification, Certified Code Generator.

Abstract

The rail transportation industry is highly dynamic and driven by the need to meet with mandatory safety criteria. Modern railway transportation systems are ever more sophisticated and rely heavily on embedded systems and communication networks. Train builders and their suppliers have to develop and deliver these increasingly sophisticated embedded systems while meeting increasingly stringent quality, safety and certification constraints.

Driven by the EN-50128 standard ‘Software For Railway Control And Protection Systems’ [3], which proposes needed methods to be used in order to provide software for control systems, train builders have applied processes and adopted tools that help to develop and validate software in compliance with safety constraints. This paper will illustrate how model based system engineering approaches and frameworks provide full business process support to:

- Validate the functional requirements of the systems (requirement management);
- Design control and safety functions at the train, vehicle and equipment levels (functional design and software architecture);
- Design and develop the software applications of electronic controllers;
- Integrate, validate and qualify electronic systems in a progressive integration process;
- Manage all changes during the lifecycle of the product (change management but also option and variant management);
- Demonstrate the compliance with the required safety integrated level,
- Train and educate drivers, rail network operators and maintenance technicians.

This kind of development and validation process and the associated tools are now successfully used by many of today’s train manufacturers and equipment providers.

Control engineers in railway commonly used development tools providing the IEC61131-3 languages ‘Programmable Controllers – Programming Languages’ [1] that are deployed in industrial automation for a long time. In this paper, we will give an overview of the languages properties and their integration in the development of software for railway.

Since the latest version of EN-50128 introduced in 2011, train control system providers are facing increased constraints on the development processes but also on the verification and validation activities. The certification of a railway control system demands ever more documentations and demonstrations that the software complies with the safety rules and criteria. All these additional tasks increase the development and validation load but are nevertheless legitimate since they are designed to ensure the safety of the product. As the railway operators ask for SIL2 product, this paper will present a new technologies helping railway OEMs and suppliers to increase the Safety Integrated Level of control software from SSIL0 to SSIL2 and to master the cost of the product.

State of The Art

In railway industry, the functions (i.e. doors, traction, brake, lighting, fire detection, air conditioning) done with conventional control system (hardware based contactors, relays, circuit breakers, etc.) are increasingly being managed by intelligent control units. Those controllers are coordinated or synchronized by a centralized and networked control system named TCMS (Train Control and Monitoring System). Depending of the safety level of the functions, the software is developed using various tools:

- softPLC providing IEC61131-3 languages
- Model-Based System Engineering frameworks (providing previous languages or data flow, state machine, etc.) and dedicated code generators;
- Hand coded in C or ADA for legacy reasons.

Some of these tools are coming from automation and control engineering (mainly for SSIL0-2 software applications) and from the avionic industry (EN50128-SSIL4 is roughly the same asDO178C-DALA) even if a product qualified for a standard is not automatically qualified for another standard).

With the growth of embedded controllers and networks in the rolling stock, train manufacturers and their sub-system suppliers need development methods and tools that are able to support the design and help stakeholder to validate requirements and functional specifications as early as possible in the project. In fact, major issues in developing these systems is the development process itself (focusing on the delivery of safety documentation) and the decreasing time and budget available for the test activities including physical tests. Like in other industries, the railway control engineers are looking for virtual test, progressive integration and qualification of the electronic control units and electro mechanical equipment.

In parallel, rolling stock and sub-system providers need to assess the overall safety of the systems, for both passenger and equipment, and to demonstrate compliance with the relevant railway rules & standards (EN-50126, EN-50128 and EN-50129 for hardware, software and product). Traditionally, the design process for embedded electronic boards is built upon a number of elementary practices and follows a V Cycle as show in *figure 1*. The EN50128 recommend the implementation of this kind of development process including verifications and validation tests.

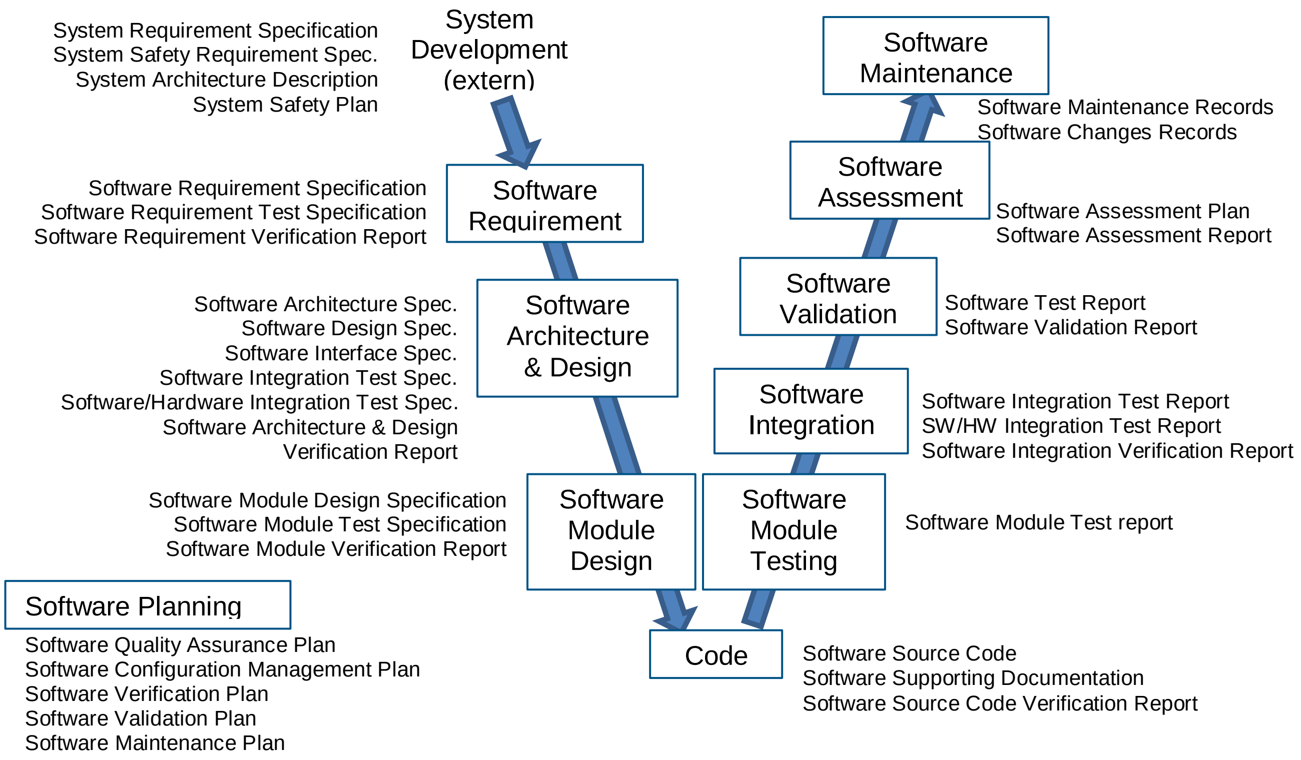

System Requirement Specification
System Safety Requirement Spec.
System Architecture Description
System Safety Plan
System Development (extern)
Software Maintenance
Software Maintenance Records
Software Changes Records
Software Requirement Specification
Software Requirement Test Specification
Software Requirement Verification Report
Software Requirement
Software Assessment
Software Assessment Plan
Software Assessment Report
Software Architecture Spec.
Software Design Spec.
Software Interface Spec.
Software Integration Test Spec.
Software/Hardware Integration Test Spec.
Software Architecture & Design Verification Report
Software Architecture & Design
Software Validation
Software Test Report
Software Validation Report
Software Integration
Software Integration Test Report
SW/HW Integration Test Report
Software Integration Verification Report
Software Module Design Specification
Software Module Test Specification
Software Module Verification Report
Software Module Design
Software Module Testing
Software Module Test report
Software Planning
Software Quality Assurance Plan
Software Configuration Management Plan
Software Verification Plan
Software Validation Plan
Software Maintenance Plan
Code
Software Source Code
Software Supporting Documentation
Software Source Code Verification Report

***Figure 1**: V-Cycle development process with documentation delivery*

Moving from a paper based development cycle…

Functional and design specifications are exclusively supported by paper based communications. The project goes slowly because the verification of textual documents requires many review iterations and exchanges between business units, central office, sub-contractors and customers. Too much time is spent in “meeting” clarification and explanation making the sentences of the specifications clear, complete and understandable by all stakeholders involved in the project. Today, too much effort is spent on “paperwork” rather than on systems design, integration and physical tests. The target is not to reduce the number and the content of paper (mandatory) but to ease this production.

The deployment of a V-Cycle methodology also introduces problem. As we can see in *figure 1*, the left side of the V-Cycle supports creation, definition, and refinement activities while the right side covers the testing activities. It means that if coding errors can be found immediately during the programming and software module testing phases, the specification errors can be detected only at the late phases of the V-Cycle during integration and validation phases. Consequently, projects frequently miss their delivery deadlines and are difficult to keep within budget. This legacy process raises important managerial issues that need to be addressed.

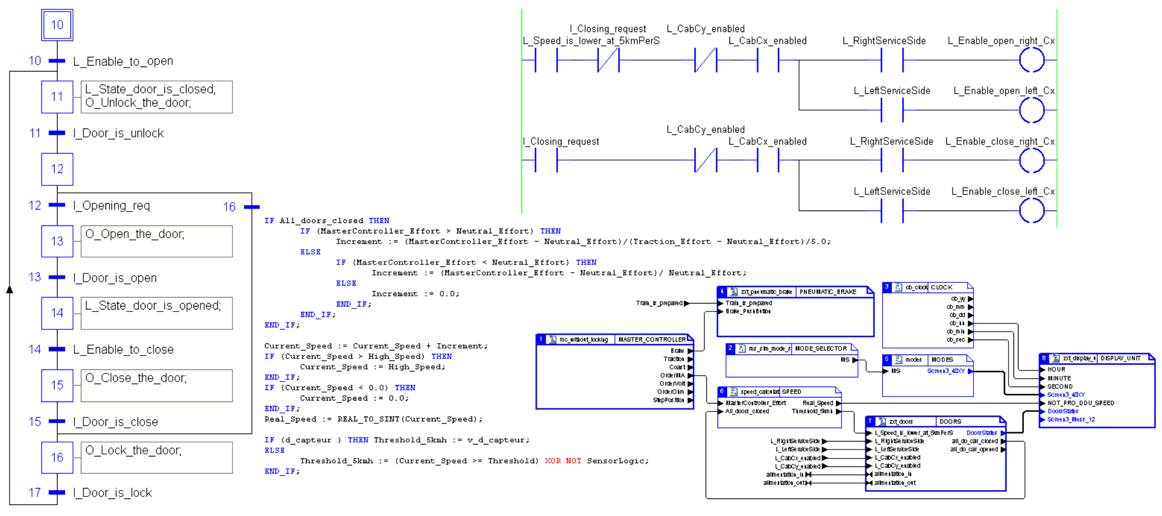
… to a Model Based System Engineering Approach

The first proposal was to move from a paper based approach to a model based system engineering approach that provides executable specifications in order to speed up the development of the control software and meet project deadlines. This process was first launched with Electronic Control Units (ECUs) for Train Control & Monitoring System projects and is today deployed on other sub systems like brake, doors or passenger information systems. For this purpose, Dassault Systèmes provides a development framework that enables the achievement of new productivity related objectives.

Model a concept, a control function, a software module

This development framework, named ControlBuild, provides a MIL (Model In the Loop) approach allowing railway engineers to specify and design a model of each control function prior to any implementation. This approach is made possible through the use of known, open and standardized languages.

In the railway industry, control engineers develop the embedded software using the languages defined within the IEC61131-3 standard for many years (see *Figure 2*). Both control engineers (including software and electrical engineers) and maintenance engineers (end users) daily use Sequential Function Chart, Structured Text, Ladder and Function Bloc Diagram for software development and modifications i.e. applying system corrections and improvements. Currently, dozens of PLC and CPU manufacturers provide coding tools based on the IEC61131-3 standard, taking the main place in the coding activities.



```
IF All_doors_closed THEN
      IF (MasterController_Effort > Neutral_Effort) THEN
            Increment := (MasterController_Effort - Neutral_Effort)/(Traction_Effort - Neutral_Effort)/5.0;
      ELSE
            IF (MasterController_Effort < Neutral_Effort) THEN
                  Increment := (MasterController_Effort - Neutral_Effort)/ Neutral_Effort;
            ELSE
                  Increment := 0.0;
            END_IF;
      END_IF;
END_IF;

Current_Speed := Current_Speed + Increment;
IF (Current_Speed > High_Speed) THEN
      Current_Speed := High_Speed;
END_IF;
IF (Current_Speed < 0.0) THEN
      Current_Speed := 0.0;
END_IF;
Real_Speed := REAL_TO_SINT(Current_Speed);

IF (d_capteur ) THEN Threshold_5kmh := v_d_capteur;
ELSE
      Threshold_5kmh := (Current_Speed >= Threshold) XOR NOT SensorLogic;
END_IF;
```

***Figure 2**: Simple models using IEC61131-3 languages*

Therefore, Control Engineers naturally improve their efficiency focusing more on design activity than on the coding phase thanks to these languages. The only requisites were that the tool has to provide:

- Code generators to ease the production of the source code of the software (in compliance with the required coding rules and properties);
- Tools and advanced features to verify, to test and to assess the models (static metrics, quality level of the model, assertions, automatic tests, code coverage, dead code identification, traceability with the requirements, etc.) as presented in the *figure 3.*

The functional design can be manually validated using scope, button, virtual panels and synoptic. Tests of the specification and design models can also be automatically executed using test procedures and functional tests features. This incremental validation approach enables time to be saved on software design and integration testing activities. The detection and the correction of an error during the system validation phases is close to 100 times more expensive than when the tests are made on the model during the design activity.

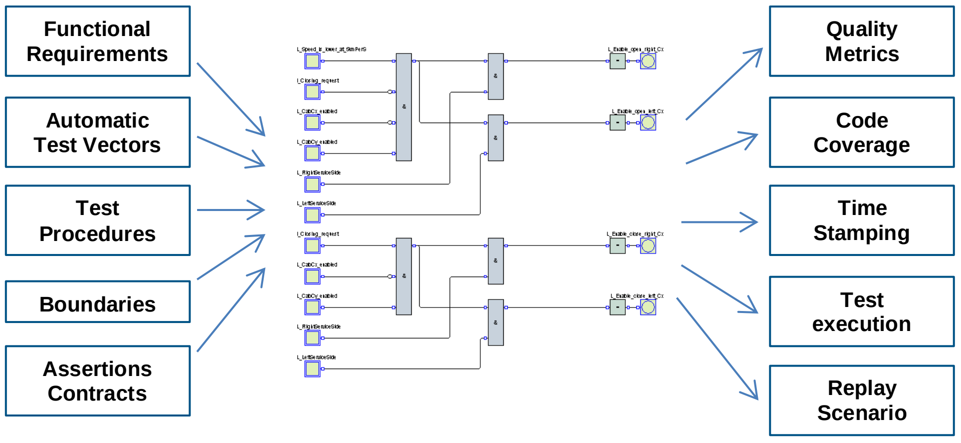


Figure 3*: Test functionalities and features*

Map the functional model on the hardware architecture

The model of the system is independent of the hardware (manufacturer, brand, operating system, etc.) and the architecture (i.e. networks and protocols). It means that we have designed the model of the software which can be deployed on one controller or distributed on many controllers. At this step of the design phase, we have to map parts (called POU - Program Organisation Unit) of the model on a defined hardware topology of the train system or sub-system. The use of the FBD (Function Block Diagram) language helps the software architect to allocate the POU on the virtual controllers (refer to *figure 4*).

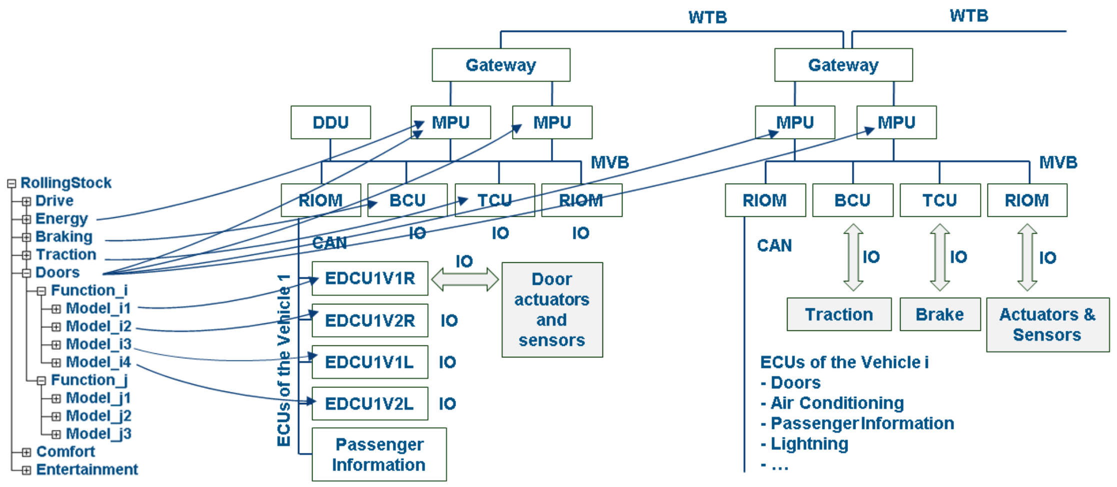


Figure 4: System Mapping – Models to Controllers

Certification of a Software Application

For railway control and protection systems, a certificate ensures that a product is compliant with the safety standards. To get a certificate, the product must be certifiable. It means that the product has been developed in such a way that it can be certified. The EN 50126 standard addresses system issues on the widest scale [2] and the EN 50129 standards addresses the approval process for individual systems [4]. Railway products embed software applications that have to be certified regarding the EN50128 standard. As in other industries, this railway standard define the notion of safety level to be achieved (SIL – generic standard, DAL for avionic, ASIL for automotive and SSIL for Railway) [5]. In avionic, the safety level determines the goal to achieve while in railway it is rather a means to implement.

SSIL – Software Safety Integrated Level

Even if the standard define 5 SSIL from 0 to 4 (4 = high level of criticality), the railway industry commonly proposes to use only 3 safety levels (technics to achieve the level 1 are closer to level 2, and level 3 is also not so far from level 4):

- SSIL0: comfort, passenger information, energy …
- SSIL2: traction/brake, doors
- SSIL4: automatic train control

To get the certification that a software application is compliant with a given SSIL, the software development team has to demonstrate to the evaluation engineers / certification authorities that:

- A well-defined software development process is used (quality assurance like ISO9001) including verification and validation tests,
- The software application covers every and only the customer requirements (using traceability requirement tools like Reqtify),
- The software application complies with the safety coding rules (like MISRA rules - Motor Industry Software Reliability Association),
- The software application is covered by test procedures,
- All associated document specification and test reports are available.

Be careful, SSIL0 doesn't mean 'no safety'; it is just the first level of safety and requires the same quality assurance process than for other SSIL. The safety case is just lighter.

Challenges and issues

Train operators are demanding even more SIL2 products for their trams, metros, inter-cities and high speed trains. It means that the Software Safety Integrated Level of the software applications of these new products have also to be compliant with SSIL2. In parallel, the latest version of the EN-50128 standard introduces more stringent activities and constraints on the development of software systems. Event for SSIL0 software application, number of companies improved their development process replacing manual programming (mainly C code) by model based design and automatic code generators.

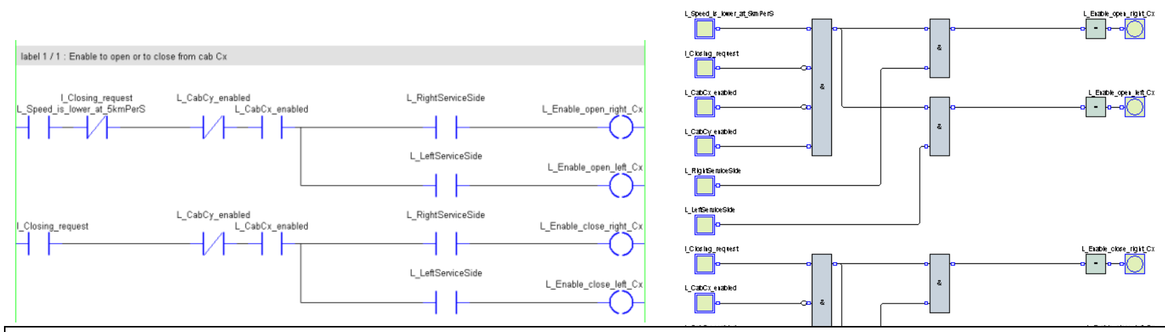
Development tools supporting the IEC61131-3 standard are the most used in the railway community of control engineers for SSIL0-2 software applications. A well-known tool used in the avionic industry, is rather used for the development of SSIL4 software for signalling - automatic train control systems – that concern less than 5% of the embedded software in a train).

An IEC61131-3 tool usually provides two kinds of code generators:

- Generation of a sequence of instructions to be executed by an interpreter. The certification is difficult to achieve because the execution machine / runtime must be certified by the tool provider. Some softPLC providers got a SIL certificate (linked to the IEC61508 generic standard) covering a limited set of instructions (AND, OR, NOT, etc.) and reserved for the execution of Boolean expressions;
- Generation of a source code to be compiled and executed on a real time platform.

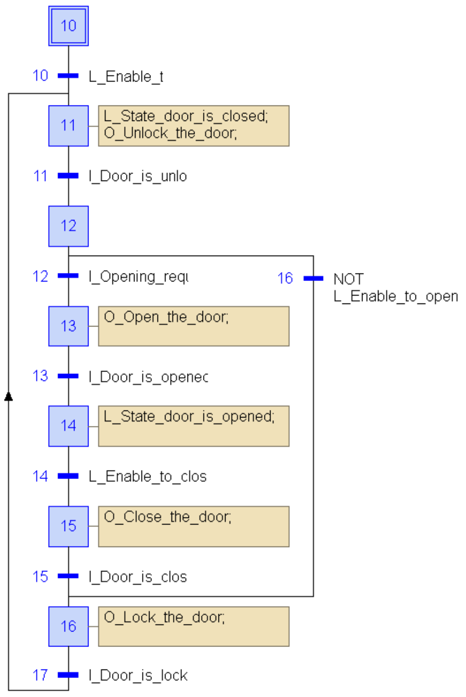
To reduce the dependency of the source code to the compiler, the EN50128 standard propose to define a subset of the language allowing a controllable execution (no jump, limited use of pointer, no dynamic memory access, no if without else, a default for switch case, only one return inside a function…).

There are many issues automatically solved because the IEC61131-3 languages are already limited (no pointer in structured text, no direct access to the operating system, etc.). Limited doesn't mean simple but sufficient to provide the code for the controller of the traction/brake systems or for the other converters, doors, air conditioning, pantograph, etc.



```
enableOpen = ((L_Speed_is_lower_at_5kmPerS && !I_Closing_request) && !L_CabCy_enabled) && L_CabCx_enabled;
L_Enable_open_right_Cx = enableOpen && L_RightServiceSide;
L_Enable_open_left_Cx = enableOpen && L_LeftServiceSide;
enableClose = (I_Closing_request && !L_CabCy_enabled) && L_CabCx_enabled;
L_Enable_close_right_Cx = enableClose && L_RightServiceSide;
L_Enable_close_left_Cx = enableClose && L_LeftServiceSide;
```

***Figure 5**: Simple example of C code generation from LD or FBD model*



```
/*pre-SFC*/
/*transitions G1*/
_Y_8 = _X_10 && L_Enable_to_open;
_Y_14 = _X_11 && I_Door_is_unlocked;
_Y_13 = _X_12 && I_Opening_request;
_Y_12 = _X_13 && I_Door_is_opened;
_Y_11 = _X_14 && L_Enable_to_close;
_Y_10 = _X_15 && I_Door_is_closed;
_Y_9 = _X_16 && (I_Door_is_locked && L_Enable_to_open);
_Y_15 = _X_12 && !L_Enable_to_open;
/*evolutions G1*/
if (_Y_8) {
    _X_10 = 0;
}
...
if (_Y_8) {
    _X_11 = 1;
}
/*timers G1*/
/*actions G1*/
if (CB_edge_bool(_X_11, _X_11_PREV) < 0) {
    L_State_door_is_closed = 0;
    O_Unlock_the_door = 0;
}
if (CB_edge_bool(_X_13, _X_13_PREV) < 0) {
    O_Open_the_door = 0;
}
if (CB_edge_bool(_X_14, _X_14_PREV) < 0) {
    L_State_door_is_opened = 0;
}
if (CB_edge_bool(_X_15, _X_15_PREV) < 0) {
    O_Close_the_door = 0;
}
if (CB_edge_bool(_X_16, _X_16_PREV) < 0) {
    O_Lock_the_door = 0;
}
```

***Figure 6**: Simple example of C code generation from SFC*

More than that, the modelling tools provide features and metrics allowing the developer to measure the complexity of the structure of the models (depth of the hierarchy, number of interfaces, number of lines, number of blocks, and number of sub-loops for example). As the IEC61131-3 standard includes textual language (ST: Structured Text), the model described in ST must also be compliant with rules such as those defined by the MISRA group.

However, at the end of the development process, the generated software application has to be certified like a manually programmed certifiable software application.

Certified SSIL2 Software generated by a Certified Code Generator

The aim of a software code generator is to help developers to remove the coding activity and to ease some verification and safety demonstration. It is why most of the development tools provide an automatic translator that takes into account a systems model and delivers source code for implementation on electronic control units.

At this step, the software engineers have to provide deliverables and demonstrations for the evaluation of the static and dynamic criteria defined in the safety standard:

- First, from a quality point of view, the generated software has to be readable, testable, verifiable and, maintainable. This means that coding rules have to be assessed as if the software were written by hand (depth of the function calls, number of overlapping 'IF' statements, number of inputs, outputs & parameters, the justification of use of pointer or global variables, dead code, out of boundary protection, defensive code, etc.).
- Another activity is to demonstrate that the proposed test cases totally cover the test requirement specification document, and then the customer requirements by using requirement traceability tools.
- Then, the software engineers have to demonstrate that the generated software has the same behaviour as the IEC61131-3 models if validation activities were already done at the design level. If not, all unit tests have to be executed and reported as if the software code were manually written.

The aim of a certified code generator is to insure that the generated code satisfy a certain level of confidence regarding the accuracy of the outputs relative to inputs. A certified code generator helps developers to remove unit tests and number of manual demonstrations.

Prerequisite: use of a well-defined input format

Since the first release of the IEC61131-3 programming standard, users want to be able to exchange their programs, libraries and projects between development environments. in fact, there are more than one hundred tools providing the IEC61131-3 textual and graphical languages but saving the description using their own specific format. Although this was not the intent of the standard itself, it was a task that the independent organization PLCopen [6] committed itself to. IEC61131-3 is focused on the software development environment.

This resulted in a workgroup named TC6 for XML. This committee defined an open interface between all different kinds of IEC61131-3 software tools. It provides the ability to transfer the information that is graphically presented on the windows of one coding tool to other coding tool. This format supports textual description (i.e. for structured text, SFC activity and receptivity) but also graphical information, like position and size of the blocks, connections (including route) between the objects of the language.

The SSIL2 Certified Code Generator

With aims of sustainability, we decided that our SSIL2 Certified Code Generator would take XML files defined by the PLCopen organization as the input and provide SSIL2 C code as the output. As in avionic, it is mandatory that the SIL level of the code generator has the same SIL level as the code it produce.

The *figure 6* shows that the SSIL2 Certified Code Generator is composed of 4 modules (light grey):

- The PLCopen XML comparator: this module verifies that the XML input file is compliant with the version of the tool.
- The IEC61131-3 library: the C functions associated to each IEC61131-3 function block have been certified by static analysis and unit tests.
- The XML to C code generator.
- The report generator of the code generation execution

These 4 certifiable software modules have been developed following a SSIL2 development, verification and validation process. The evaluation process has been executed by the certification authority named CERTIFER [7] which deliver the 8270/0157 certificate: *SILCoder version 3.00 meets the 551L2 requirements of the standard EN50128:2011*.

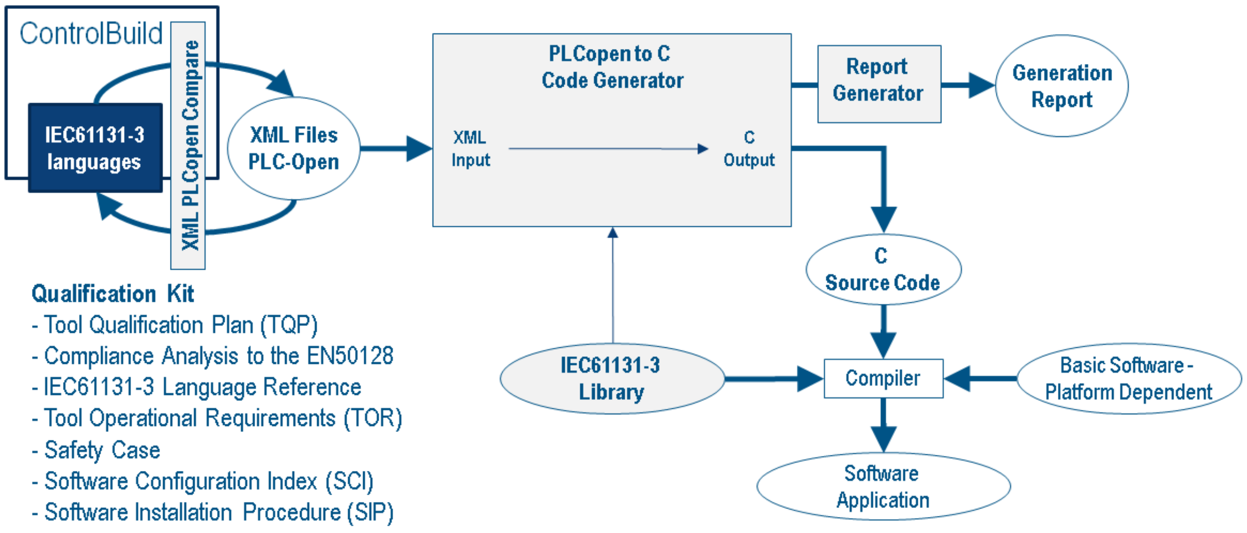

ControlBuild
IEC61131-3 languages
XML PLCopen Compare
XML Files PLC-Open
PLCopen to C Code Generator
XML Input
C Output
Report Generator
Generation Report
C Source Code
IEC61131-3 Library
Compiler
Basic Software - Platform Dependent
Software Application
Qualification Kit
- Tool Qualification Plan (TQP)
- Compliance Analysis to the EN50128
- IEC61131-3 Language Reference
- Tool Operational Requirements (TOR)
- Safety Case
- Software Configuration Index (SCI)
- Software Installation Procedure (SIP)

***Figure 6**: Certified Code Generator*

The certificate is associated to an annex which gives the list of the definition documents (requirements, plans, specifications, design, tests and reports) with the related version used for the evaluation, the results of the evaluation and also the perimeter of the certification.

Some tangibles and intangibles results

Contribution of Model based design and model based testing methodologies

A model-based development process must be used closely to the requirement management process. Modelling frameworks allows stakeholders to simulate the requirements, the functional specification, the design specification and then the software and the hardware (engineers from the automotive fully play with Model In the Loop, Software In the Loop, Hardware In the Loop and today Driver or Human In the Loop). The customer requirements and needs are immediately understood by the system provider. Functional specification can be dynamically explained to the customer, same for design, software and hardware. For example, OEM provides us return of investment showing that only 3 reviews compare to 10 before are necessary to get the design acceptance from the customer.

Executable specifications allow easy verification and validation with both internal and external non experts groups. End users like drivers or maintenance engineers are now involved in the specification reviews and design reviews rather than at the end of the project. Verification and validation at every stage of the V-Cycle significantly decreases system failure risks by finding errors at the earliest.

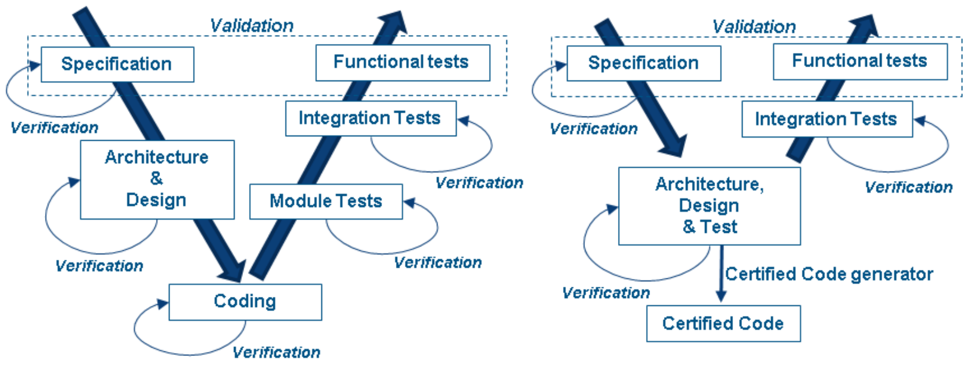


***Figure 7**: Improvement of the V-Cycle using Model based Design and Certified Code Generator*

Value of a Certified Code Generator

The SSIL2 Certified Code Generator delivers SSIL2 Certified C Code. As the software application is not just certifiable but automatically certified, software engineers can remove some demonstrations activities from their development and verification processes. The certificate gives the commitment and ensures that the software application is compliant with the SSIL2 readability and testability criteria and the recommended programming quality rules.

The software to embed in a real time controller is composed by two parts (see *figure 6*): the software application that is specific to one project and the "basic software" which makes the link between the software application and the operating system or directly the hardware if the target has not an OS.

As the specific application part is automatically certified using the Certified SSIL Code Generator, the product provider has only to certify the software dependent part of the generic target using his corporate SSIL2 Development and Verification & Validation process. In fact the certification of the basic software has to be done one time independently of the projects by the team managing the execution platform as a product: Each train maker defines and certifies a platform (hardware + basic software) that will be used during 5-10 years for multiple railway projects).

The final product is based on certified **generic** basic software (from OEM) and a certified **specific** software application (from our SILCoder). Certificates of the platform and of the code generator replace lot of tests and demonstration and save time and money. The Control System provider can focus directly on the software integration activities and software/hardware integration activities.

What we test is what we embed

How to demonstrate that the behaviour of the model has the same behaviour as the software? This big issue is solved using our IEC61131-3 development framework (ControlBuild) and our certified code generator (SILCoder). Each IEC61131-3 model is saved using the PLCopen format (native format for ControlBuild) and a unique C code is automatically generated from the XML PLCopen description. The C code of each model is generated one time and is linked with the right target platforms for simulation, code generation, debug, and deployment or test bench objectives as we can show in the *figure 8*.

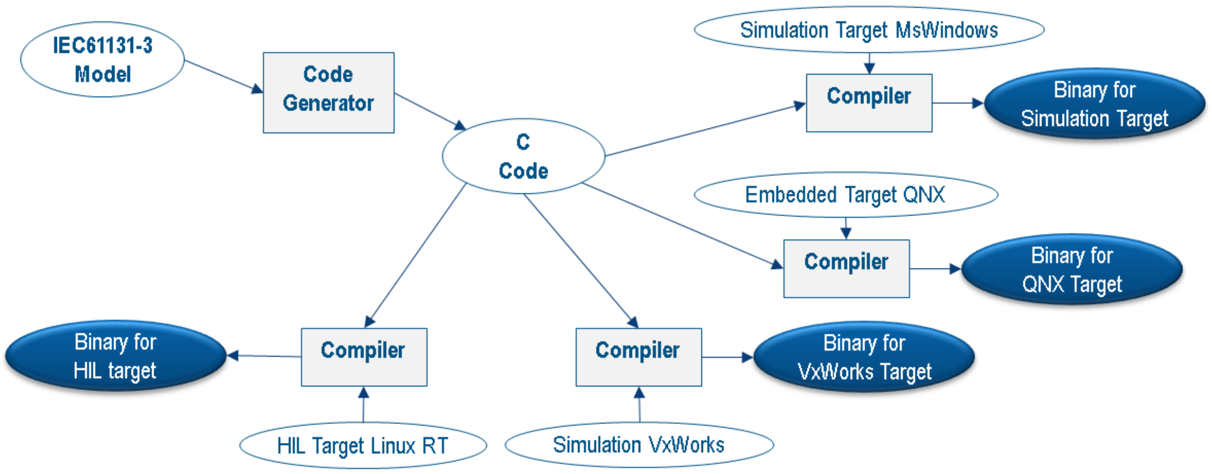


Figure 8: *What is simulated is what is embedded*

It means that we reuse the same generated C code (from the models) to provide software applications but also real time simulators (actuator, sensors and virtual controllers) for the HIL platforms (Hardware-in-the-Loop, Iron Bird in avionic). The test engineers focus on the virtual world making number of verifications. Like with Virtual Iron Bird, it is easier to execute test procedures and insert failures on the virtual environment (actuators/sensor) of the virtual product under test. Objective of virtual bench is to find and remove the last errors while the validation on HIL is to make the final demonstrations and to provide the deliverables for certification authorities.

Value for the railway industry

The IEC61131-3 languages, mainly used by the software engineers, will continue to take an ever more important place in development and delivery of software applications for railway. As the Certified Code Generator delivers the right level of safety for the control software, the gap to change the level of the certification of the software applications from SSIL0 to SSIL2 is significantly reduced. But for sure, it also means that the system provider must still upgrade his development and verification & validation process and integrate the use of modelling and code generation tools.

As in avionic with tools like SCADE®, the value of such a model based design and testing approach using ControlBuild coupled with a Certified Code Generator is promising in terms of productivity benefits and organizational effectiveness for the train makers. For the whole process, the time required to design and implement SIL2 Certified Train Control & Monitoring Systems is now halved with the ability to capitalize and reuse systems knowledge and certified assets on others projects through the standardization of train functions (and their certified products and software).

With the other IEC61131-3 tools providers, we are engaged in the official standardization of the XML PLCopen format. The name of this future standard will be IEC61131-10. This will open the use of the Certified SSIL2 Code Generator for all the developer's community. More and more products will be compliant with the SIL2 level for the safety of passengers, operators and systems.

Terminology

ASIL:	Automotive Safety Integrated Level	MVB:	Multifunction Vehicle Bus
BCU:	Brake Control unit	OEM:	Original Equipment Manufacturer
CAN:	Controller Area Network	OS:	Operating System
DAL:	Design Assurance Level	PLC:	Programmable Logical Controllers
DCS:	Digital Control Systems	PLM:	Product Lifecycle Management
DDU:	Display Driver Unit	POU:	Program organization Unit
ECU:	Electronic Control Unit	SFC:	Sequential Function Chart
EDCU:	Electronic Door Control Unit	SIL:	Safety Integrated Level
EE:	Embedded Electronic	SSIL:	Software Safety Integrated Level
FBD:	Function Block Diagram	ST:	Structured Text
HIL:	Hardware In the Loop	SUT:	System Under Test
ICD:	Interface Control Documentation	TCMS:	Train Control & Monitoring System
IP:	Intellectual Property	TCU:	Traction Control Unit
LD:	Ladder Diagram	WTB:	Wire Train Bus
MIL:	Model In the Loop	XML:	eXtended Mark-up Language

References

[1] IEC61131-3, Programmable Controllers – Programming Languages, 2001

[2] EN50126, Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS), 2000

[3] EN50128-2011, Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011

[4] EN50129, Railway applications - Communication, signalling and processing systems – Safety related electronic systems for signalling, 2003

[5] Jean-Louis Boulanger, Formal Methods – Industrial Use from Model to Code, ISTE-WILEY, 2012

[6] PLCopen for efficiency in automation, http://www.plcopen.org

[7] CERTIFER, Certificat de Type N°8270/0157 Edition 1, 2014

Merging and Processing Heterogeneous Models

P. Dissaux[1] ,B. Hall[2]

1: Ellidiss Technologies, 24, quai de la douane, 29200 Brest, France
2: Advanced Technology, Honeywell, 1985, Douglas Drive, Golden Valley, MN,USA

Introduction

Model Driven Engineering is now recognized as a way to significantly improve the development process of industrial systems and software. This approach leads to the production of various kinds of models associated to each modelling and verification step of the life cycle. All these concrete models may differ in their abstract definition (meta-model) and in their syntactic expression. Such diversity cannot be easily avoided as each modelling language brings its own specific benefit or is fundamentally associated with a particular tool or technique. However, merging and processing heterogeneous models to support all the required development activities can become a real engineering issue in the context of industrial projects.

This paper presents a solution to this problem. The proposed approach is based on the LMP[1] (Logic Model Processing) technology to provide a unique, standardized and easy to process representation of each model that is involved in a given project. Using this solution leads to the realization of a global homogeneous repository from syntactical conversion of each input model, without altering their semantic diversity. It then dramatically facilitates the development of model processing tools, such as model explorations, model verifications, model transformations and architectural reasoning.

1. The model jungle

1.1. Model roles

Models can be used at the various stages of the system development life-cycle. In the descending branch of the life-cycle, we can easily find for a single project at least requirements models, design models and a set of models associated with early verification techniques. If we make the assumption that each individual model is well defined in the context of its own purpose, the key issue is then to ensure a proper inter-operability between these various steps of the life-cycle. This implies that we can manage various kinds of model dependencies, such as traceability (e.g. between design model entities and requirement model entities), transformation (e.g. between a design model and a verification model) and consistency (e.g. between the various subsets of a design model)

1.2. Model syntax

Models can be defined with different meta-modelling languages or approaches, including BNF, XML DTDs or schemas, MOF or ECore. This choice has of course an impact on the way the concrete models can be handled in a tool or a tool-chain. Most of the time, the internal representation of a model in memory is tool dependent. On the contrary, the result of a file serialization must comply with a standard representation which depends on the corresponding meta-model syntax. For instance, BNF leads to token based human readable models, XML DTD or schema leads to XML tag based models and MOF or ECore leads to XMI tag based models. In a given project, these three kinds of concrete model syntax may have to collaborate within a same tool chain.

1.3. Model correctness

Each modelling language carries semantic concepts that are defined more or less formally by the various layers of the language definition. For token based languages, syntax compliancy brings a first level of correctness that must usually be completed by the verification of additional legality rules. With other modelling languages, more rigorous structural rules, such as relation cardinalities, can be guaranteed by construct. However, the actual correctness of a model depends on its foreseen usage, and two models described with the same modelling language may have different contents according to their applicative domain. For instance, the model of a real-time system may look correct in the scope of the verification of its static architecture but not correct in terms of its timing behaviour. Another source of discrepancy may come from the compliancy with the corporate or project engineering rules. For instance the way to build a system in SysML[2] may significantly vary between the various users or tool vendors.

1.4. Model inter-operability

All these differences may be fully justified but can become a blocking issue for the definition of complete tool-chains or for system wide model integration. Several approaches may be considered to solve these model inter-operability issues. The first natural solution is to express all the models with the same meta-modelling language and implement all the tools within the same framework. The typical example of this solution is the use of the Eclipse platform as it has been done with the Topcased and Polarsys[3] initiatives. Such an “all-in-one” approach does minimize the inter-operability problem but raises a certain number of other issues like a lack of modularity and an increase of the effort required to include specialized legacy models and tools. Another solution consists in offering a standardized communication layer by the mean of a logical bus that can be used to ensure model transformations and tool interaction in a transparent way. An example of this solution is ModelBus[4] which allows for heterogeneous tools interaction, but is dedicated to ECore based models. The alternate approach that we are describing below does not have these restrictions. This approach is called Logic Model Processing (LMP).

2. Logic Model Processing

LMP is based on the use of the Prolog[5] language to formally specify rules to be applied to an appropriate representation of the applicative model. This representation of the model is composed of Prolog facts. Prolog (Programmation Logique) is a declarative language that is used to express rules applying on predicates. Rules can then be combined using Boolean Logic. Prolog syntax is very simple and most programs can be specified using AND, OR and NOT logical operators.

LMP consists of a methodology, a set of tools and prolog libraries.

Model Driven Engineering activities are supported by LMP as follows:
- The meta-model classes define prolog fact specifications whose parameters names correspond to the attributes names of the classes.
- An instantiated model consists in a populated prolog facts base, where facts parameters values correspond to classes attributes values.
- The model processing program is expressed as a set of prolog rules whose predicates are others rules or facts.
- To execute a LMP program, it is necessary to produce the facts base associated with the model to be processed, to merge it with the rules base associated with the processing to be performed and to run a query with the prolog interpreter.

With the prolog language being an ISO standard, any prolog environment can be used to support the LMP approach. The one that has been used until now is sbprolog[6]. Other environments such as SWI-Prolog[7] are also being considered. One of the particularities of sbprolog is that the facts and rules bases can be described in textual form or in a binary form (byte code). Sbprolog binary files can be concatenated which highly facilitates the realization of modular processing features and the merge of input models.

In addition to the prolog interpreter itself and its run-time environment, a set of additional tools and reusable libraries are also part of the LMP tool-box. These include in particular input model parsers and output model un-parsers (or printers). The currently available parsers are for XML/XMI models (xmlrev), AADL[8] models (aadlrev), and programming languages. Facts base generation can also be implemented for memory stored models handled by modeling tools.

The main benefits brought by the LMP approach are:
- A clear separation between the model to be processed (facts base) and the model processing program (rules base).
- A strong traceability between model processing requirements and their implementation (one rule per requirement).
- The declarative and logical programming style offered by the prolog language.
- The ability to define modular set of processing rules and to link them together at run time.
- The ability to use a same implementation language for all kinds of model processing, i.e. navigation within the model language constructs (query language), verification of model properties (constraint language), model to model, model to text and text to model transformations (transformation language).

3. Current LMP Applications

3.1. Stood

LMP principles have been applied in their early phase in the Stood[9] design tool. It has been used in this context for more than twenty years to implement various features such as static rules checkers, code and documentation generators as well as reverse engineering tools.

One of the most significant successes of the use of the LMP technology within the Stood tool has been the qualification of customized model verification by Airbus in support of the DO 178 certification process.

3.2. AADL Inspector

After the positive experience of the use of the LMP approach for increasing the capabilities of the Stood tool, it was decided to apply it in an extensive way for the development of the AADL Inspector[10] framework.

AADL Inspector is a model processing environment that can parse AADL models and connect them to a variety of verification and generation tools, such as Cheddar[11] for scheduling analysis, Marzhin[12] for event based simulation and Ocarina[13] for source code generation. AADL Inspector can also be used to convert SysML or UML-MARTE[14] models into a corresponding AADL specification to take advantage of the existing connections with the processing tools.

AADL Inspector can thus be seen as a generic model processing framework using AADL as pivot language and LMP as transformation technique.

3.3. The TASTE tool-chain

The TASTE[15] tool-set resulted from spin-off studies of the ASSERT project, which started in 2004 with the objective to propose innovative and pragmatic solutions to develop real-time software. One of the primary targets was satellite flight software, but it appeared quickly that their characteristics were shared among various embedded systems.

The solutions that have been developed now comprise a process and several tools. The development process is based on the idea that real-time, embedded systems are heterogeneous by nature and that a unique UML-like language was helping neither their construction, nor their validation. Rather than inventing yet another “ultimate” language, TASTE makes the link between existing and mature technologies such as Simulink, SDL, ASN.1, C, Ada, and generates complete, homogeneous software-based systems that one can straightforwardly download and execute on a physical target.

Within the TASTE tool-chain, LMP is used to insure a link between the Domain Specific Models (Interface View and Deployment View) and the corresponding AADL specification that is used during the model processing phases (real-time analysis and code generation).

3.4. The PMM editors

The Property Model Methodology[16] is a system engineering approach that involves several interconnected models: the Specification Model, the Property Based Requirements, four kinds of Design Models and the overall System Model.

A graphical editor is being developed to support this modelling approach and the use of LMP is foreseen to insure the various required model processing needs in this context.

4. Using LMP to process heterogeneous models

The examples of use of the LMP technology that have been presented in the previous section are all confined in the scope of a particular tool or tool-chain: processing HOOD[17] models in Stood, processing AADL models in AADL Inspector and in TASTE.

However, one of the most interesting benefits of this approach is that it can easily be generalized to address the models-interoperability issue that has been expressed in section 1 of this paper. In this section, we explain how LMP can be used to convert, merge and process heterogeneous models.

4.1. Converting heterogeneous models

The role of model parsing in the LMP context consists of performing a syntactic transformation from the original model stored in memory or serialized in a file into a normalized prolog facts base.

In the case of text based input models, the conversion are insured by a parser which output consists of a list of prolog predicates.

In the case of memory based input models, which are for instance produced by a graphical tool, the prolog predicates must be generated with a dedicated printing or serialization feature.

When the sbprolog environment is used, LMP provides a C library for the production of binary predicates. This library can be linked to the model parsers or editors.

4.2. Merging heterogeneous models

The conversion step that is described in the previous paragraph can be applied to all the input models, whatever meta-model they comply with and whatever they are serialized in a file or stored in memory.

The portable way of merging converted models is to concatenate the elementary textual prolog facts bases that have been produced by each conversion tool. I can be noted that additional information can be inserted at that stage under the direct form of dedicated prolog facts. This may be especially useful to introduce processing instructions (pragmas) into the merged facts base.

The main constraint that applies at this stage is the management of facts redundancy and ordering. A good practice to avoid issues is to ensure that each conversion tool produces a different set of predicates. When this is not possible, the variation of the number of parameters (arity) can be used to avoid facts overwriting. Another solution consists of using a dedicated parameter to identify the model source.

In the case of sbprolog, it is not required to take care of the way textual facts are ordered. With other prolog environments it may be mandatory to group all the similar facts together. Moreover, sbprolog binary facts files can be also concatenated.

At this stage, the resulting facts base is a homogeneous data repository representing the merged heterogeneous input models that is ready for any kind of processing.

4.3. Processing heterogeneous models

Most of what has been explained for the input models facts base can also be applied to the model processing rules bases. The main difference is that the rules bases are usually statically defined and stored in a model processing library, whereas the facts bases are dynamically elaborated from the current state of the applicative model. Another difference comes from the facts/rules separation that has strictly been applied until now for all the LMP realizations. However, some future applications may require that rules are also specified within the input model. This may be the case for instance while adding model constraints insertions during the modelling phases.

All the variety of processing can now be applied to the merged facts base, such as static rules checkers, dedicated model generators to feed verification tools and source code generators.

4.4. Example of use

This section presents a practical example of use of the approach. This case study has been simplified as much as possible for illustrative purpose. We are addressing what could be the development process for a library of operations on complex numbers.

We consider a development workflow that would be composed of four separate steps. Each step is associated with an ISO 12207 standard activity and is likely to use different description languages and tools:

- step1: System Requirements Analysis (5.3.2).
- step2: Software Requirements Analysis (5.3.4).
- step3: Software Architectural Design (5.3.5).
- step4: Software Coding and Testing (5.3.7).

4.4.1 System requirements analysis

The table below gives a small subset of possible requirements for the realisation of a mathematical library on complex numbers.

Id	Name	Text
1	R_ComplexLib	The complex number library must define the complex number type and operations on complex numbers.
2	R_ComplexType	A complex number type must have a Real part and an Imaginary part.
3	R_ComplexAttributes	Real and Imaginary parts of a complex number must be real numbers.
4	R_ComplexAdd	The two operands and the return value of the add operation must be complex numbers.
5	R_ComplexSub	The two operands and the return value of the sub operation must be complex numbers.

When done with a requirements analysis tool, such as IBM Doors or SysML compliant editors, a requirements model can be serialised in various formats. For the purpose of the example, we have selected the Requirements Interchange Format (ReqIF), as it is an OMG standard. A small fragment of the corresponding file is given below:

```
<SPEC-OBJECTS>
  <SPEC-OBJECT LONG-NAME="R_ComplexLib" ...
  <SPEC-OBJECT LONG-NAME="R_ComplexType" ...
  <SPEC-OBJECT LONG-NAME="R_ComplexAttributes"
  <SPEC-OBJECT LONG-NAME="R_ComplexAdd" ...
  <SPEC-OBJECT LONG-NAME="R_ComplexSub" ...
</SPEC-OBJECTS>
```

Applying the appropriate LMP parser (xmlrev) to this file provides the corresponding list of prolog facts:

```
isXMLTag('#39','SPEC-OBJECTS','#13','39').
isXMLTag('#40','SPEC-OBJECT','#39','40').
isXMLAttribute('#40','SPEC-OBJECT',
       'LONG-NAME','R_ComplexLib','40').
...
```

The fact type **isXMLTag/4** keeps track of the XML tags hierarchy whereas **isXMLAttribute/5** provides the name and value of each attribute for each XML tag. This low level description may be hard to use during the next steps. We can thus improve the access to relevant information by introducing a first level of processing that defines a new set of predicates of type **isSpecObject/1**:

```
getSpecObjects :-
  isXMLTag(X,’SPEC-OBJECTS’,_,_),
  isXMLTag(Y,’SPEC-OBJECT’,X,_),
  isXMLAttribute(Y,_,’LONG-NAME’,R,_),
  assert(isSpecObject(R)).
```

This rule creates a new list of facts that hides the syntactic complexity of the original XML structure and filters the needed data for further use, such as performing requirements traceability.

```
isSpecObject('R_ComplexLib').
isSpecObject('R_ComplexType').
isSpecObject('R_ComplexAttributes').
isSpecObject('R_ComplexAdd').
isSpecObject('R_ComplexSub').
```

4.4.2 Software requirements analysis

We assume that the next modelling step consists in formalizing the data type requirements thanks to a UML class diagram, as shown below:

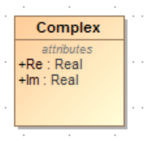


Figure 1 : a UML class

This UML class diagram can be serialized in standard XMI format. A little fragment of the resulting .uml file is given below:

```
<uml:Model xmi:type='uml:Model' ...>
<packagedElement xmi:type='uml:Class' ...
 name='Complex'>
  <ownedAttribute xmi:type='uml:Property' ...
  name='Re'/>
  <ownedAttribute xmi:type='uml:Property' ...
  name='Im' />
</packagedElement>
```

We can then use the same LMP parser as in the previous step to convert this UML model into an equivalent list of prolog facts.

```
isXMLTag('uml#3','uml:Model','#2','3').
isXMLAttribute('#3','uml:Model',
       'xmi:type','uml:Model','3').
isXMLTag('#5','packagedElement','#3','5').
isXMLAttribute('#5','packagedElement',
       'xmi:type','uml:Class','5').
isXMLAttribute('#5','packagedElement',
       'name','Complex','5').
isXMLTag('#6','ownedAttribute','#5','6').
isXMLAttribute('#6','ownedAttribute',
       'xmi:type','uml:Property','6').
isXMLAttribute('#6','ownedAttribute',
       'name','Re','6').
...
```

The XMI serialisation generated by UML tools may be huge even for a small model. To pre-select the useful information for a given processing purpose, we can create a more specialised facts base from the original one by specifying a filtering rule:

```
getUmlClasses :-
  isXMLTag(X,'uml:Model',_,_),
  isXMLTag(Y,'packagedElement',X,_),
  isXMLAttribute(Y,_,'xmi:type','uml:Class',_),
  isXMLAttribute(Y,_,'name',C,_),
  assert(isUmlClass(C)),
  isXMLTag(Z,'ownedAttribute',Y,_),
  isXMLAttribute(Z,_,'xmi:type',
                        'uml:Property',_),
  isXMLAttribute(Z,_,'name',P,_),
  assert(isUmlProperty(C,P)).
```

The new facts base would then looks like the following, and could be easily enriched to contain other details such as attribute types or requirement satisfy abstractions.

```
isUmlClass('Complex').
isUmlProperty('Complex','Re').
isUmlProperty('Complex','Im').
```

4.4.3 Software architectural design

We are now considering the software architecture and express the library as an AADL package, so that it can be used by the other parts of the application. Although the AADL standard has a graphical notation, its main usage is with the textual notation that is human readable and scalable.

```
PACKAGE ComplexLib
PUBLIC
WITH Base_Types;

  DATA Complex
  END Complex;

  DATA IMPLEMENTATION Complex.others
  SUBCOMPONENTS
    Re : DATA Base_Types::Float;
    Im : DATA Base_Types::Float;
  END Complex.others;

  SUBPROGRAM Add
  FEATURES
    C1 : IN PARAMETER ComplexLib::Complex;
    C2 : IN PARAMETER ComplexLib::Complex;
    R : OUT PARAMETER ComplexLib::Complex;
  END Add;

  SUBPROGRAM Sub
  FEATURES
    C1 : IN PARAMETER ComplexLib::Complex;
    C2 : IN PARAMETER ComplexLib::Complex;
    R : OUT PARAMETER ComplexLib::Complex;
  END Sub;

END ComplexLib;
```

In order to be able to process an AADL specification with LMP, we need to use the AADL parser (aadlrev). The result of the parsing is the facts base that is shown below:

```
isComponentType('ComplexLib','PUBLIC',
  'Complex','DATA','',6).
isComponentImplementation('ComplexLib','PUBLIC',
  'Complex','others','DATA','','',9).
isSubcomponent('ComplexLib','Complex','others',
  'Re','DATA','Base_Types::Float','','',10).
isSubcomponent('ComplexLib','Complex','others',
  'Im','DATA','Base_Types::Float','','',11).
isComponentType('ComplexLib','PUBLIC',
  'Add','SUBPROGRAM','',15).
isFeature('PARAMETER','ComplexLib','Add','C1',
  'IN','','ComplexLib::Complex','','',16).
isFeature('PARAMETER','ComplexLib','Add','C2',
  'IN','','ComplexLib::Complex','','',17).
isFeature('PARAMETER','ComplexLib','Add','R',
  'OUT','','ComplexLib::Complex','','',18).
...
```

Due to the token based nature of the AADL syntax as opposed to the XML/XMI based languages, the resulting facts base is much more compact and directly usable without having to define a new set of more precise predicates.

4.4.4 Coding

For the last step of our development process, we will process an implementation of the library in Ada language. A possible realisation in source code could be:

```
package ComplexLib is
  type Complex is record
    Re : Float;
    Im : Float;
  end record;
  function add (
    C1 : IN Complex;
    C2 : IN Complex )
    return Complex;
  function sub (
    C1 : IN Complex;
    C2 : IN Complex )
    return Complex;
end ComplexLib;
```

We can then use another parser of the LMP toolbox (adarev) to build a facts base from the Ada code.

```
packageSpec('ComplexLib','_root_').
typeComponent('ComplexLib','Complex',
  'Re','Float','').
typeComponent('ComplexLib','Complex',
  'Im','Float','').
typeSpec('ComplexLib','Complex','...').
operationSpec('ComplexLib','add','...').
param('ComplexLib','add','...',
  'C1','Complex','in','').
param('ComplexLib','add','...',
  'C2','Complex','in','').
param('ComplexLib','add','...',
  'return','Complex','out','').
...
```

For the same reason as for AADL, there is no need to create a new facts base to select the useful information.

4.4.5 All together

Although they are coming from different modelling languages, the syntactic transformation into prolog predicates allows for global processing of the merged model.

The facts sub-bases can be concatenated to provide all the required data to perform cross-activity processing. In particular, this can be used to verify the consistency of the workflow, such as requirements coverage or code compatibility with the architecture.

All these verification rules can be expressed in standard prolog language, as shown in the two simple examples given below.

Rule1. the data types specified in the software specification must be defined in the software architecture:

```
checkR1 :-
 isUmlClass(T),                           /*UML*/
 not(isComponentType(_,_,T,'DATA',_,_)),/*AADL*/
  write('Error R1 for: '), write(T).
```

Rule2: the data types specified in the software architecture must be defined in the source code:

```
checkR2 :-
  isComponentType(_,_,T,'DATA',_,_),    /*AADL*/
  not(typeSpec(_,T,_)),                 /*Ada*/
  write('Error R2 for: '), write(T).
```

Similar rules could be defined to check that all the system requirements are properly covered by design entities.

After having shown how LMP could ease static processing of merged heterogeneous models, we will now consider dynamic architectural reasoning.

5. Architectural Reasoning Using LMP

The prolog fact-based representation of the LMP form presents a good foundation for the logical reasoning and processing of the architectural models. In the next sections we present some simple examples to illustrate the flexibility of the LMP approach.

5.1.Physical Separation and Independence Analysis

To illustrate this potential, we present a simple LMP-based extension to illustrate how physical zonal independence can be assessed from the LMP model.

In modern aircraft, there is often a need to ensure that the independence assumed within a fault-tree is sufficient to mitigate the potential of physical damage that may arise from adverse system events such as fire, or explosions. Many manufactures require a minimum physical separation among redundant elements. In Integrated Modular Avionic (IMA) architectures, assuring that this separation is achieved for all of the hosted functions can be non-trivial, as multiple sub-function elements are distributed across the IMA processing and input/output hardware elements. As architectures become increasing networked and distributed, the complexity of such analysis may only increase, hence the ability to address it systematically is attractive.

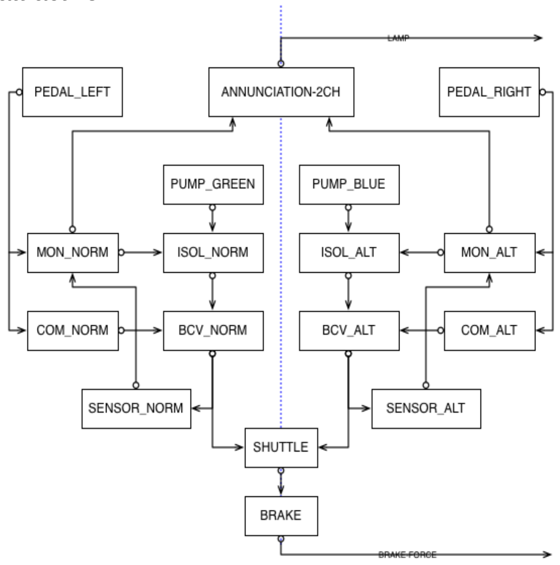


Figure 2 : Example Wheel Brake System

To illustrate the technique we present the wheel brake system case study, shown in Figure 2. This system comprises two independent hydraulic circuits that are each controlled by a dual lane command /monitor computation system. For each circuit, the command processor modulates the expected pressure to achieve the required braking force; the monitor processor supervises the commanded braking operation, monitoring commanded output pressure sensor feedback. If the monitored pressure is not in agreement with the monitor's expected limits, the monitor closes the isolation valve and removes all hydraulic pressure, rendering the channel in active. It then yields control to the other channel.

For brevity of the presentation, our example analysis focuses on the placement of the command and monitoring processing hardware of each lane.

To introduce the physical location of components onto the model a new AADL property is defined. In our simple example, we utilize a single point, but in practice this may be a series of points, which represent the physical boundaries of the components. Inherently extensible, via property sets adding such notions to an AADL model is very straightforward.

```
property set Location  is

Location: list of record
( x_pos : aadlreal;
  y_pos :aadlreal;
  z_pos : aadlreal; )
applies to
( processor, system, abstract, device );

end Location;
```

To represent both good and bad configurations, two system implementations are defined, by extending the base implementation. In the first configuration the command and monitor components within each lane are placed to be adjacent, and the computation elements of the Alt and Norm lanes are separated by 10 meters, as illustrated below.

```
System implementation
wbs_com_mon_dual_lane.good_impl
  extends wbs_com_mon_dual_lane.impl

properties
Location::Location =>
  ([x_pos => 1.0; y_pos => 1.0; z_pos=> 1.0;])
  applies to monAlt;
Location::Location =>
  ([x_pos => 1.1; y_pos => 1.0; z_pos=> 1.0;])
  applies to comAlt;
Location::Location=>
  ([x_pos => 10.0; y_pos => 1.0; z_pos=> 1.0;])
  applies to comNorm;
Location::Location =>
  ([x_pos => 10.1; y_pos => 1.0; z_pos=> 1.0;])
  applies to monNorm;

end wbs_com_mon_dual_lane.good_impl;
```

In the second configuration the monitor of the Alt channel is swapped with the monitor of the Norm channel. It should be noted that, this configuration is insufficient with respect to the required physical channel separation, since failure of a single physical zone will destroy critical components of both lanes of redundancy. Hence, it is our intention is to illustrate this using model analysis.

The AADL composite error model, records the assumptions of the structure of the redundancy, hence the first stage of analysis is to convert this into LMP form. This is done using the ADDL parser (aadlrev) that has been extended under this work to capture the structure of the Error Annex logical expressions.

```
composite error behaviour
states
--Unannunciated braking loss
[
( ( pumpGreen.Failed or isolNorm.Failed or
    bcvNorm.Failed or monNorm.Failed or
    comNorm.Failed ) and
  ( pumpBlue.Failed or isolAlt.Failed or
    bcvAlt.Failed or monAlt.Failed or
    comAlt.Failed ) ) or
( pedalLeft.Failed and PedalRight.Failed ) or
brake.Failed
]-> UnannunciatedBrakingLoss;
```

The LMP representation for the above model comprises two facts; **isEMV2CompositeStateElem** is used to store failure conditions, and **isEMV2CompositeStateExpr** facts are then used to map the logical relationships of the elements. Two example facts are shown below.

```
isEMV2CompositeStateExpr('wbs','wbs_com_mon_dual
_lane','impl','unnamed_A1','unnamed_K1','$1','pu
mpGreen.Failed','OR','isolNorm.Failed',235).

isEMV2CompositeStateElem('wbs','wbs_com_mon_dual
_lane','impl','unnamed_A1','unnamed_K1','$2','bc
vNorm.Failed','NIL',235).
```

In a similar manner to other LMP parsers, element and expression identifies and indexes '$1' are maintained to support the mapping and relating of the facts. Once in LMP form, the next stage of the physical separation, is to convert the logical structure of the fault-tree into a form that can be executed with prolog, This conversion comprises a simple mapping. Each expression is converted to a dynamic prolog fact as illustrated below.

```
 init :-
        dynamic('pumpGreen.Failed'/0),
        dynamic('isolNorm.Failed'/0),
        dynamic('bcvNorm.Failed'/0),
        dynamic('monNorm.Failed'/0),
        dynamic('comNorm.Failed'/0),
        dynamic('pumpBlue.Failed'/0),
        dynamic('isolAlt.Failed'/0),
        dynamic('bcvAlt.Failed'/0),
        dynamic('monAlt.Failed'/0),
        dynamic('comAlt.Failed'/0),
        dynamic('pedalLeft.Failed'/0),
        dynamic('PedalRight.Failed'/0),
        dynamic('brake.Failed'/0).
```

The **init** predicate introduces all of the potential faults as dynamic facts in the prolog database. It is complimented by a clear, predicate (not shown) that retracts all facts related to failures. Clear is then used to establish a clean baseline for iterative analysis and queries. This is also useful to support working in the interactive prolog shell, when manually exploring failure combinations.

The **isEMV2CompositeStateExpr** are similarly reduced and mapped to a simplified executable form as illustrated below.

```
exp11 :- exp5,exp10.
exp12 :- 'pedalLeft.Failed','PedalRight.Failed'.
exp1 :- 'pumpGreen.Failed';'isolNorm.Failed'.
exp2 :- exp1;'bcvNorm.Failed'.
exp3 :- exp2;'monNorm.Failed'.
exp4 :- exp3;'comNorm.Failed'.
exp6 :- 'pumpBlue.Failed';'isolAlt.Failed'.
exp7 :- exp6;'bcvAlt.Failed'.
exp8 :- exp7;'monAlt.Failed'.
exp9 :- exp8;'comAlt.Failed'.
exp14 :- exp11; exp13.
exp15 :- exp14;'brake.Failed'.
exp5 :- exp4.
exp10 :- exp9.
exp13 :- exp12.
```

Logical structure of these is generated from the structure of the composite error model state-annotations. It should be noted that the code and effort required to generate executable form from the LMP representation is very small comprising less than 100 lines of prolog in total.

We remind that the logical operator AND (resp. OR) is expressed in prolog by a comma (resp. a semicolon).

Once in this reduced form, prolog is able to compute how the combination of failures impacts the top-level event. In our simple the top-level event is represented by the expression with the highest index.

The next stage of the analysis is to generate the set of failures that can correspond to the different zones. Once processed by LMP, the location properties introduced previously yield a set of facts for each component as shown below.

```
isRecordField('wbs','wbs_com_mon_dual_lane',
'good_impl','monAlt','LOCATION::LOCATION',1,'x_p
os','1.0',275).
isRecordField('wbs','wbs_com_mon_dual_lane',
'good_impl','monAlt','LOCATION::LOCATION',1,'y_p
os','1.0',275).
isRecordField('wbs','wbs_com_mon_dual_lane',
'good_impl','monAlt','LOCATION::LOCATION',1,'z_p
os','1.0',275).
```

The local facts are then grouped by component using a **ftacomponent** predicate as shown below, that simply maps the component name and X,Y,Z location.

```
ftacomponent(P,T,I,Name,X,Y,Z) :-
  isRecordField(P,T,I,Name,'LOCATION::LOCATION',
    1,'x_pos',XA,_),
  isRecordField(P,T,I,Name,'LOCATION::LOCATION',
    1,'y_pos',YA,_),
  isRecordField(P,T,I,Name,'LOCATION::LOCATION',
    1,'z_pos',ZA,_),
  atom_number(XA,X),
  atom_number(YA,Y),
  atom_number(ZA,Z).
```

The system then examines all instances of the **ftacomponent** and generates zones of collocated component using the a simple separation auxiliary predicate, that returns true if the two components are located within a defined separation limit (**Distance**), which in our case was 6 meters.

```
separation((X1,Y1,Z1),(X2,Y2,Z2),Distance):-
  Xd = X2 - X1,
  Yd = Y2 - Y1,
  Zd = Z2 - Z1,
  D is sqrt((Xd * Xd) + (Yd * Yd) + (Zd * Zd)),
  D < Distance.
```

As illustrated below the code to **build_zones** is relatively terse comprising only a few lines of code. This is one of the attractions of the declarative nature of the prolog processing.

```
build_zones([],PP,TT,II,Acc,Acc).
build_zones([(_,X,Y,Z)|T],PP,TT,II,Acc,Final):-
findall((B,BX,BY,BZ),
   (ftacomponent(PP,TT,II,B,BX,BY,BZ),
   (seperation((X,Y,Z),(BX,BY,BZ),6))),Zone),
    (not(member(Zone,Acc)) ->
       append(Acc,[Zone],NewAcc);
       NewAcc = Acc ),
    build_zones(T,PP,TT,II,NewAcc,Final).
```

Once the zones are complete, we simply instantiate a zonal related failure set for each zone, but adding zone predicates to the output. The zone predicates for the good and bad wheel brake command monitor configurations are shown below.

```
zone_good_impl0:-
       assertz('comAlt.Failed'),
       assertz('monAlt.Failed').
zone_good_impl1:-
       assertz('monNorm.Failed'),
       assertz('comNorm.Failed').
```

```
zone_bad_impl0:-
       assertz('monNorm.Failed'),
       assertz('comAlt.Failed').
zone_bad_impl1:-
       assertz('comNorm.Failed'),
       assertz('monAlt.Failed').
```

To explore the system, it is only necessary to instantiate each of the generated zonal fault-sets with the fault tree expression logic discussed previously, and querying the state of the top-level hazard, in out case **exp15**. To package such analysis in smaller form large LMP database, a stand-alone file 'ftap.pro' is generated by the LMP processing logic.

This can then be loaded into the interactive SWI-Prolog environment. A trace from an interactive session following the loading of the file is shown below for the good component placement example.

```
- ['ftap.pro'].
true.
?- init.
true.
?- clear.
true.
?- exp15.
false.
?- zone_good_impl0.
true.
?- exp15.
false.
?- zone_good_impl1.
true.
?- exp15.
true .
```

Evaluating **exp15** following the single zone failure **zone_good_impl0** concludes false, indicating that the failure of **zone_good_impl0** is not sufficient to cause the top-level event. However, the subsequent additional failure of **zone_good_impl1** does cause the top-level event, and the subsequent query of **exp15**.

Repeating the procedure with the bad configuration, we see that failures of the single zone are sufficient to cause the failure of the top-level event as illustrated in the trace below.

```
?- ['ftap.pro'].
true.
?- init.
true.
?- clear.
true.
?- exp15.
false.
?- zone_bad_impl0.
true.
?- exp15.
true .
```

Note, that in our toy example, we are using a simplified representation of location. In real deployments our single point may be expanded to present the component boundary points. Similarly, other types of queries such as the use of a common CPU type, cooling zone, and or power-supply distribution etc., are easily implemented given AADL's extensible property provisions. In each case basic analysis technique would remain largely unchanged, only requiring adaption of the **build_zones** criteria.

5.2. Modelling Completeness Checks.

A second application of the LMP processing is the implementation of automated model completion and completeness checking. In large systems the level of detail and abstraction within the model needs to be managed and maintained. Organizations often maintain modelling standards that define the required model content. However, enforcing such standards can be cumbersome without the appropriate automation.

However, if the model is expressed in LMP, the automation of consistence and compliance checks becomes very simple. Given that, all aspects of the architecture are represented by facts, simple queries against the fact bases can be generated for each requirement. For example, as shown below, only a few lines of code are necessary to execute the query to check that all processing hardware components have a consistent error model associated with them.

```
isComponentType(P,_,X,'PROCESSOR',_,_),
not(isAnnex(P,X,_,_,'EMV2',_,_)),
writeErrorMessage(P,X).
```

By examining the architectural model, using additional system composition predicates and/or predicates derived from fault-tolerance theory predicates, the architectural correctness may be simply validated. For example, a simple predicate may check that all bus components have a specified error model, e.g. Bit Error Rate (BER). A more advanced predicate may check that protocols that are bound to the bus utilize a suitable end-to-end data transport protocol to mitigate the expected error-rate. More elaborate queries, such as Byzantine vulnerability analysis, are also possible, by highlighting all instances where forked data paths of the logical model, have terminals that bind to different physical components.

Related Work

The fault-tree representation used by our illustrative case-study was inspired from the original work of Shuchi[18].

In the area of architecture model processing, several alternate solutions have been explored, such as REAL [19], RESOLUTE [20] and AGREE [21]. The definition of a “constraint annex” is also under discussion by the AADL standardisation committee.

On-going and Future Work

LMP is used for the development of the new processing plug-ins that will be integrated in the future distributions of the AADL Inspector tool. The current work is focused on extending the import capabilities for UML profiles models such as MARTE, SysML, SCADE System, or CAPELLA as well as new processing features, especially for safety analysis.

In order to facilitate the connection with Domain Specific Modelling Languages, an automatic generation of most the prolog rules that are required to parse, navigate and process ECore based models is being developed. A similar approach is also envisaged for XSD definitions.

Following the idea that LMP allows a tool agnostic infrastructure to be developed, integration with solutions like Modelbus is also considered.

Another area of active research is the generation/ derivation of the AADL Error-Annex composite model annotations from a case-based reasoning approach of the known component failure modes in conjunction with architectural topology.

Conclusion

This paper introduces the raising issue of heterogeneous models processing and proposes an original solution to address it. This solution merges the principles of logic programming and those of model driven engineering to define the Logic Model Processing (LMP) approach.

This approach and the supporting tools are described in the paper and several examples are provided to illustrate its benefits.

References

[1] “Model Verification: Return of Experience”, P. Dissaux and P. Farail, ERTS 2014.
[2] SysML: Systems Modeling Language, http://sysml.org
[3] Polarsys: https://www.polarsys.org/
[4] Modelbus: https://www.modelbus.org/
[5] prolog: ISO/IEC 13211-1, 1995
[6] sbprolog: Stony Brook Prolog, https://www.cs.cmu.edu/Groups/AI/lang/prolog/impl/prolog/sbprolog/0.html
[7] SWI-Prolog: http://www.swi-prolog.org/
[8] AADL: SAE AS-5506B, 2012: http://www.aadl.info/
[9] Stood: http://www.ellidiss.fr/public/wiki/wiki/stood
[10] AADL Inspector: http://www.ellidiss.fr/public/wiki/wiki/inspector
[11] Cheddar: http://beru.univ-brest.fr/~singhoff/cheddar/
[12] “The SMART Project: Multi-Agent Scheduling Simulation of Real-time Architectures”, P. Dissaux, O. Marc and all, ERTS 2014.
[13] Ocarina: http://www.openaadl.org/ocarina.html
[14] MARTE: Modeling and Analysis of Real-Time and Embedded Systems, http://omgmarte.org/
[15] TASTE: http://taste.tuxfamily.org/
[16] “Model Based System Engineering”, P. Micouin, ISTE and John Wiley & Sons editors, September 2014.
[17] HOOD: Hierarchical Object Oriented Design, http://www.esa.int/TEC/Software_engineering_and_standardisation/TECKLAUXBQE_0.html
[18] "Construction of a fault tree using prolog." Fukuda, Shuichi. ICF6, New Delhi (India) 1984. 2013.
[19] “Modeling and verification of memory architectures with AADL and REAL”, S. Rubini, F. Singhoff and J. Hugues, ICECSS 2011.
[20] "Resolute: an assurance case language for architecture models." Gacek, Andrew, et al. Proceedings of the 2014 ACM SIGAda annual conference on HILT. ACM, 2014.
[21] "Compositional verification of architectural models.", Cofer, Darren, et al. NASA Formal Methods. Springer Berlin Heidelberg, 2012. 126-140.

Session 18

Process

Thursday 28th, 15:00 – Guillaumet

SAVOIR: Reusing specifications to favour product lines

SAVOIR Advisory Group
represented by
Jean-Loup Terraillon
European Space Agency
Savoir@esa.int

Keywords
SAVOIR, space avionics, product line, reference specifications

Abstract
SAVOIR has taken inspiration from AUTOSAR, although the underlying industrial business model is different. The space community is smaller, the production is based on a few spacecraft per year, and there are industrial policy constraints. Still, there is a need to streamline the production of avionics and improve competitiveness of European industry. Reference architectures, reference specifications and standard interfaces are an efficient mean to achieve the goal. Space agencies and space industry are actively working at developing such reference specifications. Reusing specification is expected to allow reusing products.

1. Description of SAVOIR

A. What is SAVOIR

SAVOIR (***S**pace **A**vionics **O**pen **I**nterface a**R**chitecture)* (http://savoir.estec.esa.int) is an initiative to federate the space avionics community and to work together in order to improve the way that the European Space community builds avionics subsystems.

The objectives are:

- to reduce the schedule and risk and thus cost of the avionics procurement and development, while preparing for the future,
- to improve competitiveness of avionics suppliers,
- to identify the main avionics functions and to standardise the interfaces between them such that building blocks may be developed and reused across projects
- to influence standardisation processes by standardising at the right level in order to obtain equipment interchangeability (the topology remains specific to a project).

- to define the governance model to be used for the products, generic specifications, interface definition of the elements being produced under the SAVOIR initiative.

The process is intended to be applied as part of the Agencies ITTs, and throughout the subsequent procurements and development process. A particular goal is to have SAVOIR outputs exploited in future projects and relevant products as part of European supplier's portfolios.

SAVOIR is coordinated by the SAVOIR Advisory Group (SAG) including representative of ESA, CNES, DLR, AirbusDS, TAS, OHB, RUAG, Selex Galileo, and Terma.

SAVOIR has been presented in ERTSS2012 [ERTSS2012]. The purpose of this paper is to give an update on the status of the initiative, the new technical areas that have been investigated, the documents that are available, the review process aiming at achieving consensus on the documents, and the application in space projects.

It is interesting to note how SAVOIR intentions have evolved. The initial ambition was to propose not only reference architecture and specification, but also building block products allowing the construction of the avionic systems. The discussions with industry have shown that it was more efficient to leave the development of the products to industry and let them manage their product lines. However, it was clear that the role of the Agencies was to prepare generic specifications at system level, as well as the basic technology that industry will then use to build products.

In addition, SAVOIR started with three pillars (data handling, control systems and software). It appeared soon that the operability was another important pillar which has a substantial impact on the variability of on-board avionics. This addresses space ground interface, operability concepts, and can also lead to harmonizing the software architecture on board and on ground.

B. State of the Art

The closest example of state of the art is AutoSar [AUTOSAR], triggered in the automotive industry. The business model is different (producing high number of cars and equipment), and therefore the resulting cooperation consensus is different. But the principle of "cooperating on standards and competing on implementation" remains. This was the motivation to start SAVOIR, but some differences where quickly noted:

- Managing complexity is the driver for AutoSar, but not for SAVOIR. Instead, SAVOIR aims at product lines within a domain of reuse which is relatively well known. The concern was more the variability of the requirements than their complexity. Beside, complexity is intended to be addressed with the model based approach, both system and software.
- The hardware architecture includes nearly a hundred of ECUs in a car, whereas the spacecraft platform computers can be counted with two hands' fingers.
- The business model is different, while millions of the same car are sold, space industry delivers some one-off spacecrafts a year. Instead, the European space cooperation rules enforce sub-contracting through "geographical return" (ESA must finance the industry of a Member State proportionally to the Member State's contribution to ESA), creating the need for interoperability and clean-cut architecture.

The American initiative SUMO (Space Universal MOdular Architecture) [SUMO] has been started by the American office of the director of national intelligence, with inspiration of SAVOIR. According to the SUMO published data (credit Bernie Collins ODNI/AT&F), this innovative satellite acquisition has for objective to reduce the overall cost of space assets to government clients, and to enhance the global responsiveness of the space industrial base.
The policy drivers are based on the US National Space Policy 2010: "To promote a robust domestic commercial space industry, foster fair and open global trade and commerce through the promotion of suitable standards and regulations that have been developed with input from U.S. industry."

Other worldwide initiatives include (credit SUMO):

- SPA: AFRL's Space PnP Architecture – focused on reducing satellite development to months instead of years. A draft standard has been created through AIAA. It evolved to MONArch.
- cFE: NASA Goddard's core Flight Executive software framework enables basic software functions to be reused across programs, while allowing for tailoring of mission-specific software application functionality.
- Common Avionics Architecture (SpaceAGE Bus): NASA Goddard combines cFE/CFS with modular hardware (intra-box electrical & mechanical) definition for board level building-block functional elements; may be combined to form box level functionality
- FDK: DARPA's F6 Developer's Kit, which is a set of open source interface standards, protocols, behaviours, and reference implementations thereof, necessary to develop a new module that can fully participate in a fractionated cluster.
- Joint Architecture Standard (JAS): DOE Sandia National Labs – satellite PL processing & data com architecture, focuses on increasing mission flexibility, accommodating enhanced sensor performance, optimizing payload size, weight & power (SWaP) consumption.

2. Scope of the project

A. Initial working groups

At the time of ERTSS2012, the SAVOIR Advisory Group was supported by three sub working groups:

SAVOIR-SAIF (Sensor/Actuator InterFace)

This working group addresses the electrical interface of the sensor and actuators used for the attitude control and the guidance of the spacecraft.
The group achievement has been to detect the excessive costs due to the systematic redevelopment of an RS422 protocol, and therefore to propose its standardization in the scope of ECSS. A preliminary study defined physical and data link layer requirements as input to an update of the ECSS-E-ST-50-14C standard. This update is on-going.

SAVOIR-SAFI (Sensor/Actuator Functional Interface)

This working group is in charge of the standardization of the functional interface with sensors and actuators.
The group's achievement has been to define a standardized functional interface for the Star Trackers, which was proposed to be included in the ECSS-E-ST-60-20C (Star Sensor Terminology And Performance Specification).

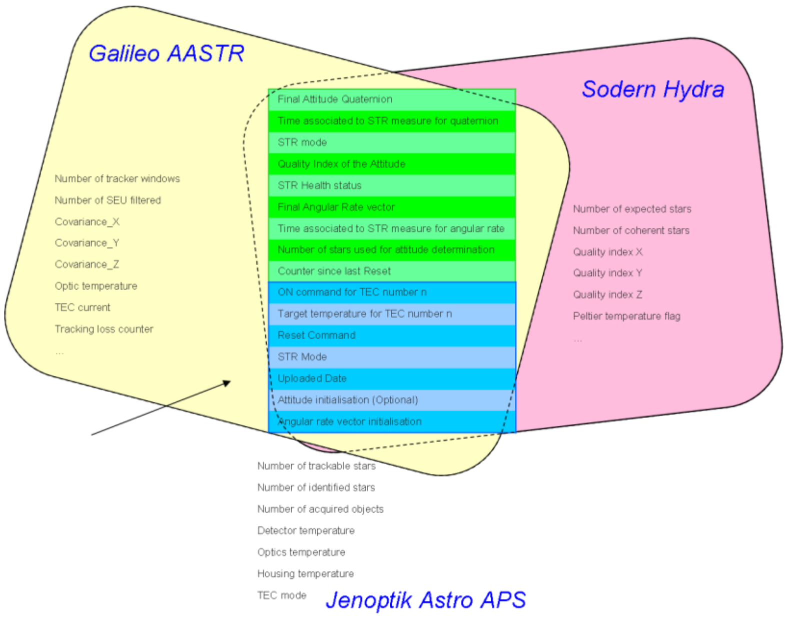

Galileo AASTR
Sodern Hydra
Jenoptik Astro APS
Final Attitude Quaternion
Time associated to STR measure for quaternion
STR mode
Quality Index of the Attitude
STR Health status
Final Angular Rate vector
Time associated to STR measure for angular rate
Number of stars used for attitude determination
Counter since last Reset
ON command for TEC number n
Target temperature for TEC number n
Reset Command
STR Mode
Uploaded Date
Attitude initialisation (Optional)
Angular rate vector initialisation
Number of tracker windows
Number of SEU filtered
Covariance_X
Covariance_Y
Covariance_Z
Optic temperature
TEC current
Tracking loss counter
...
Number of expected stars
Number of coherent stars
Quality index X
Quality index Y
Quality index Z
Peltier temperature flag
...
Number of trackable stars
Number of identified stars
Number of acquired objects
Detector temperature
Optics temperature
Housing temperature
TEC mode

Figure 1 Star Tracker functional interface

The group investigated then the possibility to expand the same activity to gyroscopes. The results of the discussion in SAFI are that there are coarse and FOG gyro which are so different that it is not possible to harmonize between the two families. So there could be two standardizations. However, the FOG gyros have been harmonized already by the primes who are procuring them. The coarse gyros are too few for harmonization. Therefore this investigation was stopped. For the other sensors-actuators, which are simpler, the standardization is not necessary. Therefore the work of SAFI was stopped.
Further activity on sensor-actuator interface will happen around the technology of Electronic Data Sheet (EDS) that intends to support the automatic configuration of the access to the devices within the software architecture.

SAVOIR-FAIRE (Fair Architecture and Interface Reference Elaboration)

This working group is in charge of the on-board software reference architecture.
The group achievement has been to come with a complete definition of the on-board software reference architecture [OSRA], including:

- Users' needs and high level requirements for such an architecture
- the definition of two layers (application and execution platform)
- a Space Component Model description to support the description of application with components
- a functional specification of the execution platform, and of the interface between the components and the execution platform.
- a demonstrator of feasibility in ESA laboratory.

B. Recent working groups

Since ERTSS 2012, two new working groups have been created:

SAVOIR-IMA (Integrated Modular Avionics)

The working group addresses the spin in of the aeronautical Integrated Modular Avionics (IMA) concept into space.

The group's achievement has been to produce high level requirements for a concept of Time and Space Partitioning in the software architecture, use cases and their prioritization, the definition of industrial roles in the production of IMA for Space systems, and an architecture of a TSP based execution platform.

Several R&D activities supported this achievement. The last one intended to harmonize the "classical" architecture produced by SAVOIR-FAIRE with the IMA architecture. Its results are now being integrated in the final SAVOIR-FAIRE documents.

The motivation was to segregate the integration, verification and validation of some software functions that could be independent, such as the various payload software or sensor software, while running them on a single computer and thus saving electronics costs.

SAVOIR-MASAIS (MAss Storage Access Interfaces and Services)

The working group addresses the data storage (for platform and payload) and the related use of files in spacecraft's operation.

The group achievement for the moment is to produce system level requirements for a data storage system. Later results will be to produce requirements of a File Management System, supported by a specific R&D activity. Spacecraft operation has traditionally be based on packet transfer, and only recently files are used on-board (e.g. Euclid). This requires harmonization between the spacecraft operators and the on-board implementation of data storage.

A new working group is under creation:

SAVOIR-UNION (User Needs In On-board Network , name TBC)

The working group will investigate the definition of the functional, performance, operational and interface requirements of the functional links and their management. The scope is limited in the identification and characterisation of the needs of users in term of communication and does not address the definition of communication standards and protocols.

The group will come with system level requirements of an avionics network. The difficulties encountered in some project to integrate and validate some avionics link within the system composed of the main computer, the data storage and the payloads, has shown that the avionics system approach had not been enough investigated.

3. Major results

A. Documents

SAVOIR delivers two levels of documents: the system level documents used by the customer Agencies, and the product specification used by primes in their procurement.

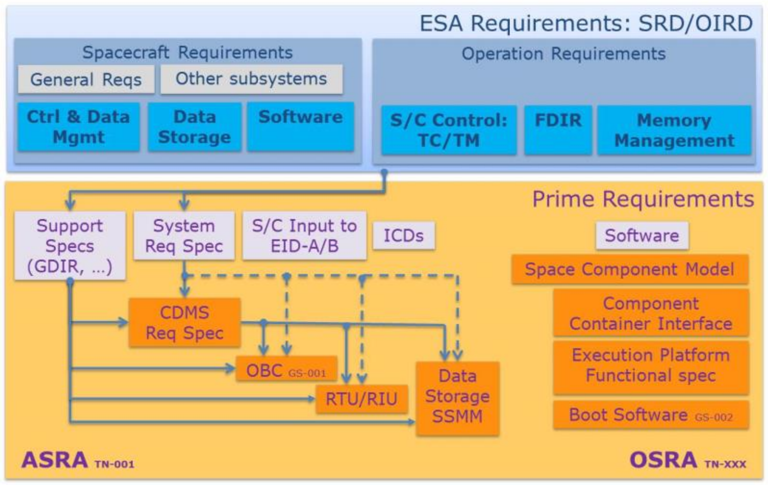


Figure 2 SAVOIR documentation tree

In Figure 2, the top documents are the ESA System Requirement Documents (SRD) and Operation Interface Requirement Documents (OIRD) that will be developed in the frame of SAVOIR (for avionics) and more generally at Agency level for other disciplines. The bottom documents are the product specifications intended to be used by the Large System Integrators. They address hardware (ASRA side: Avionics System Reference Architecture) and software (OSRA side: On-board Software Reference Architecture).

In addition to the reports of some of the working groups (SAVOIR-FAIRE, SAVOIR-IMA, SAVOIR-SAIF and SAVOIR-SAFI), 4 documents are available at the time of ERTSS2016:

- SAVOIR documentation tree (SAVOIR-TN-000 technical note)
- Avionics System Reference Architecture (SAVOIR-TN-001 technical note). This document introduces the reference architecture with a functional approach. A list of the functions usually implemented in hardware on a spacecraft platform (and partially on payload) is provided, together with their description.
- On-Board Computer generic specification (SAVOIR-GS-001 applicable document). This document list the requirements applicable to the group of functions commonly implemented in an on-board computer.
- Initialisation Sequence Software (boot software) (SAVOIR-GS-002 applicable document). This document lists the minimum set of requirements that are applicable to any boot software of a spacecraft processor.

The two –GS- documents (Generic Specification) associate requirements with a formal reference, their justification, and some notes on their applicability. They can be optional or mandatory. They are associated with parameters to be defined when the document is actually used in a project. They are intended to be later administered in an IBM Rational DOORS requirement database.

The way the documents are distributed is based on the European Space Software Repository (ESSR), a repository intended for the diffusion of ESA software assets. This repository allows for a controlled distribution, in particular allowing the dissemination within member states only. The ESSR has been opened to the public in September 2015.

SAVOIR status is also disseminated every year in the Avionics Data Control Software Systems in October at Estec (http://adcss.esa.int).

Out of SAVOIR, ESA is working on generic System Requirements Document and Operation Interface Requirement Document that will harmonize further spacecrafts procurements.
However, within SAVOIR, some sections of the avionics part of the SRD have been internally drafted. A draft generic OIRD has been produced in ESA and is under SAG review.

In the future, the software documents are being refined in order to be ready to enter into a public review:

- Space Component Model (including pseudo-component definition and component-container interface) (applicable)
- Execution Platform functional specification (applicable)
- Interaction layer – execution platform interface (technical note)
- Execution Platform internal interface (technical note)

B. Public review

The process of public review is similar to the ECSS process. Eurospace, the entity that represents the space industry, has nominated reviewers from selected companies. An organisation note of this industrial consultation has been agreed. The public review started in January 2015. About 500 comments (159 majors and the rest minors) from 20 companies were received on the 3 documents. The spirit of the review was very constructive. The penetration of the SAVOIR knowledge within industry was considerably improved. About 300 modifications on the 3 documents were implemented intending to improve the understanding of the documents (scope, applicability, and glossary), the applicability of the document (refinement of the domain of reuse, applicability matrix, additional industrial relevance) and some technical aspects of the document. The review used the web-interface tool that ESA actually deploy in the spacecraft's project reviews, which allows for a full visibility of the comments by the community and a full traceability of the evolutions.

C. Reference Architecture

The SAVOIR Advisory Group, supported by R&D activities, came up with a reference avionics architecture:

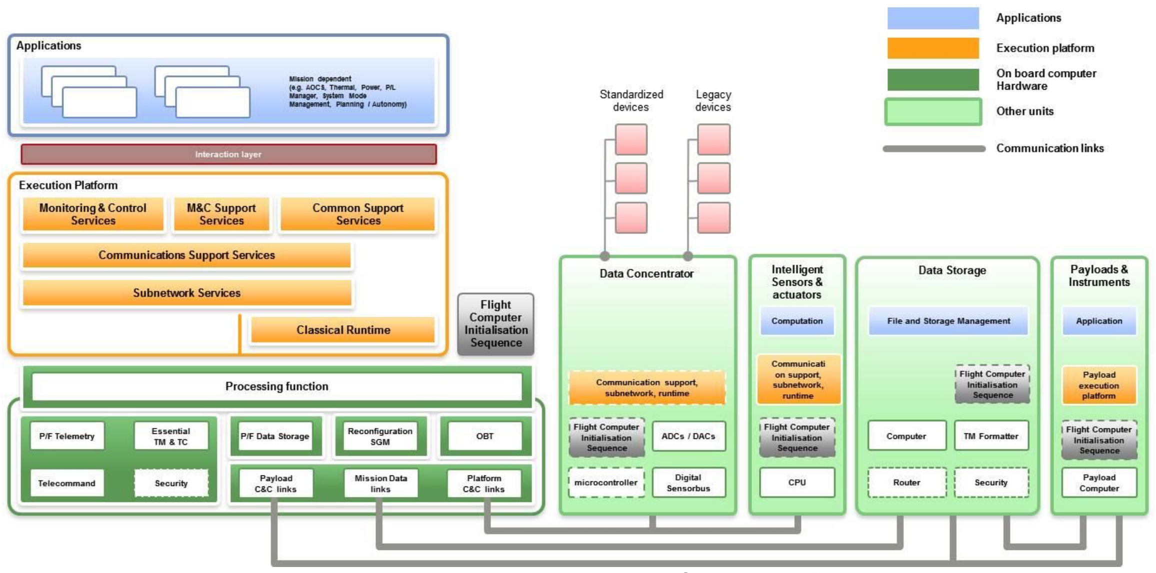

Applications
Execution platform
On board computer Hardware
Other units
Communication links
Applications
Mission dependent (e.g. AOCS, Thermal, Power, P/L Manager, System Mode Management, Planning / Autonomy)
Interaction layer
Execution Platform
Monitoring & Control Services
M&C Support Services
Common Support Services
Communications Support Services
Subnetwork Services
Classical Runtime
Flight Computer Initialisation Sequence
Processing function
P/F Telemetry
Essential TM & TC
Telecommand
Security
P/F Data Storage
Reconfiguration SGM
OBT
Payload C&C links
Mission Data links
Platform C&C links
Standardized devices
Legacy devices
Data Concentrator
Communication support, subnetwork, runtime
Flight Computer Initialisation Sequence
ADCs / DACs
microcontroller
Digital Sensorbus
Intelligent Sensors & actuators
Computation
Communication support, subnetwork, runtime
Flight Computer Initialisation Sequence
CPU
Data Storage
File and Storage Management
Flight Computer Initialisation Sequence
Computer
TM Formatter
Router
Security
Payloads & Instruments
Application
Payload execution platform
Flight Computer Initialisation Sequence
Payload Computer

Figure 3 SAVOIR Avionics System Reference Architecture

On the hardware side, avionics was organized in functional blocks with interfaces:

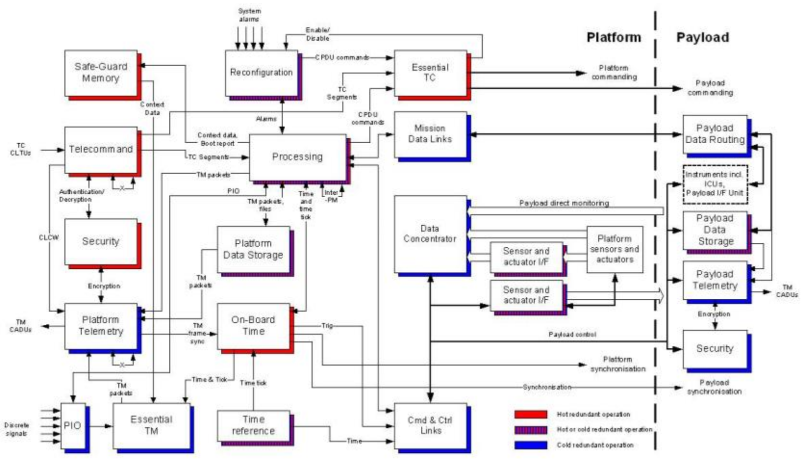
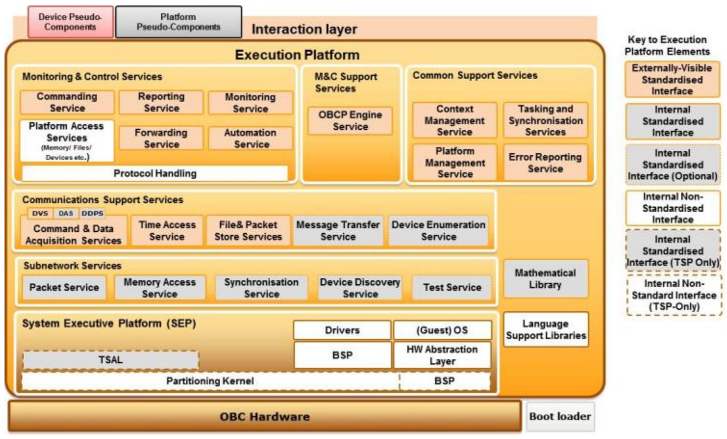


Figure 4 The avionics functions

The mapping of these functions on actual physical boxes is let to industry to decide, in the scope of their definition of product lines. Some examples are given in the document SAVOIR-TN-001.

On the software side, the notion of execution platform was further described in the following diagram:



Device Pseudo-Components
Platform Pseudo-Components
Interaction layer
Execution Platform
Monitoring & Control Services
Commanding Service
Reporting Service
Monitoring Service
Platform Access Services (Memory/ Files/ Devices etc.)
Forwarding Service
Automation Service
Protocol Handling
M&C Support Services
OBCP Engine Service
Common Support Services
Context Management Service
Tasking and Synchronisation Services
Platform Management Service
Error Reporting Service
Communications Support Services
DVS
DAS
DDPS
Command & Data Acquisition Services
Time Access Service
File& Packet Store Services
Message Transfer Service
Device Enumeration Service
Subnetwork Services
Packet Service
Memory Access Service
Synchronisation Service
Device Discovery Service
Test Service
Mathematical Library
System Executive Platform (SEP)
Drivers
(Guest) OS
BSP
HW Abstraction Layer
TSAL
Partitioning Kernel
BSP
Language Support Libraries
OBC Hardware
Boot loader
Key to Execution Platform Elements
Externally-Visible Standardised Interface
Internal Standardised Interface
Internal Standardised Interface (Optional)
Internal Non-Standardised Interface
Internal Standardised Interface (TSP Only)
Internal Non-Standard Interface (TSP-Only)

Figure 5 Details of the Execution Platform

D. Progress

Key indicator of progress are the assiduity to the SAG meetings and the constant high audience of the dissemination events ADCSS, as well as the increasing maturity of the SAVOIR documents, through substantial R&D work, prototyping, industry exchanges and reviews.

The main difficulty encountered is to initiate this change process. SAVOIR is a not only a technical adaptation, but it is a "life style" involving view point change from all the stakeholders. Agencies have to specify always the same way, Large System Integrators have to procure with the same specification, and Suppliers have to arrange for product lines. Harmonizing the way that ITTs are done at Agencies level is a substantial task, as well as it is to influence product line management in large companies who have also commercial markets out of the institutional Agencies market.

The applicability of SAVOIR documents must also be defined. Indeed, if they are labelled in ITTs as Reference Document, there is no incentive for industry to take them on-board projects, and there is no way for the SAVOIR coordination to measure their suitability and to adapt the documents to the needs. On the other hand, if they are labelled as Applicable Documents, they are binding, traceability of non-compliance is feasible and can be fed back to the SAVOIR organization. But at the same time, non-compliance is seen at contractual level as a competitive disadvantage decreasing the chances to win the contract, whereas the nature of non-compliance to SAVOIR is not going to change the mission performance, but maybe to achieve the mission objectives with a different set of functional specification. Still, it will break the product lines industrial objective.

The measure of SAVOIR success is somehow difficult to quantify, as long as the documents are not formally made applicable. However, they find their way in some projects, and several proposals have clearly been derived from the SAVOIR documents.

4. Conclusion

The objectives fulfilment is assessed in the following way.

The two first objectives are very long term and cannot be directly measured:

- to reduce the schedule and risk and thus cost of the avionics procurement and development, while preparing for the future,
- to improve competitiveness of avionics suppliers,

The other objective starts to be reached:

- to identify the main avionics functions and to standardise the interfaces between them such that building blocks may be developed and reused across projects

This has been done and examples of building blocks are coming (OBC, RTU, software operating system, execution platform)

- to influence standardisation processes by standardising at the right level in order to obtain equipment interchangeability (the topology remains specific to a project).

A number of standards have been created or modified, in particular in ECSS. The SAVOIR technical activities have also been useful to review external standards, in particular the [CCSDS]. The [SOIS] standards have been reviewed by industry, and consequently their use in OSRA has been better targeted. The [MOS] standard have been analysed and a technical roadmap has been produced for a long term consideration in on-board architecture.

- to define the governance model to be used for the products, generic specifications, interface definition of the elements being produced under the SAVOIR initiative.

We are at the beginning of the governance of products, but software experience exists with the management at ESA level of the qualification of the operating system [RTEMS]. In the same line, discussions are on-going to define the governance of the separation kernel [Xtratum]. Cooperation ESA CNES is discussed around the product [LVCUGEN].

Some lessons learned may be derived from the exercise:

- it takes a lot of time to federate a community around objectives that are globally beneficial,
- however, this background continuous harmonization between customers and suppliers is extremely efficient to keep heads aligned in the same direction.
- customers should not try to rule it all, instead each stakeholder can act at his level in the scope of its own constraints
- avionics is not only software, hardware and control, but also operability has a substantial impact on it.

SAVOIR should not be seen as direct product standardization, where on-the shelves products are imposed by the agencies in spacecrafts. This would not work. Instead, SAVOIR is a continuous Harmonization process within the avionics community where each stakeholder adjusts his behaviour at his level for the benefit of all industry. This change process is challenging, but the progressive penetration of the concept within the many layers of the avionics community is successfully on-going.

5. Literature

[AUTOSAR] (http://www.autosar.org)

[CCSDS] The Consultative Committee for Space Data Systems (http://public.ccsds.org/default.aspx)

[ERTSS2012] What You Must KNOW About SAVOIR… SAVOIR Advisory Group represented by J.L. Terraillon, February 2012

[ESSR] https://essr.esa.int/

[MOS] Mission Operations Services (http://public.ccsds.org/publications/archive/520x0g3.pdf) standard.

[OSRA] Documentation : SAVOIR-HB-001 i1 r0 - SAVOIR On-board Software Reference Architecture Training Material
(https://essr.esa.int/ ; registration needed; access to ESA member states representatives only)

[RTEMS] Real-Time Executive for Multiprocessor Systems (https://www.rtems.org/).
[Xtratum] (http://www.xtratum.org/).

[SOIS] The Spacecraft Onboard Interface Services Area (http://public.ccsds.org/publications/SOIS.aspx)

[SUMO] Space Universal MOdular Architecture (SUMO): CCSDS Spring Plenary Bordeaux, France
http://www.google.nl/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=2&ved=0CCIQFjABahUKEwjkicLXo7LIAhWFVhoKHaOPCXg&url=http%3A%2F%2Fcwe.ccsds.org%2Fsois%2Fdocs%2FSOIS-APP%2FMeeting%2520Materials%2F2013%2FSpring%2FSUMO%2520CCSDS%2520Spring%2520Plenary.pdf&usg=AFQjCNEnk7inVj4QS-onNXkMhUmbglKQUQ&sig2=wH9enbnuDDiwYp79UBKWkA

[LVCUGEN] Logiciel de Vol Charge Utile GENérique (https://indico.esa.int/indico/event/53/session/10/contribution/53/material/1/0.pdf).

A Lean Systems Engineering Approach for the Development of Safety-critical Avionic Systems

Ralf Bogusch, Sabine Ehrich, Roland Scherer, Tobias Sorg, Robert Wöhler
Airbus Defence and Space, Claude-Dornier-Straße, 88039 Friedrichshafen, Germany
{ralf.bogusch, sabine.ehrich, roland.scherer, tobias.sorg, robert.woehler}@airbus.com

***Abstract* — The strong cost pressure of the market and rigorous safety regulations affect the development of avionic systems. Safety standards like SAE ARP4754A and RTCA DO-178C require high efforts for assuring compliance with applicable airworthiness requirements. Hence, industry is forced to continuously optimize their lifecycle processes and tool environments to facilitate the development of safety-critical systems. In this paper, we report on our experience of adopting lean enablers to systems engineering. The approach covers requirements quality analysis, model-based systems engineering, model-based testing, product family engineering and safety analysis. The experiences gained from an industrial case study in the aerospace domain.**

***Keywords* — Application lifecycle management, domain ontologies, end-to-end traceability, functional analysis, lean systems engineering, model-based systems engineering, model-based testing, product family engineering, requirements patterns, requirements quality analysis, safety analysis, variants.**

1. INTRODUCTION

Safety-relevant avionic systems are becoming more and more complex. Additionally, safety standards like SAE ARP4754A and RTCA DO-178C require rigorous development and assurance activities to demonstrate compliance with applicable airworthiness requirements. However, defects introduced in system specifications such as missing or unclear requirements may cause rework in the downstream activities leading to unnecessary costs. In addition, product variants sharing similar features are often developed by different teams hindering the reuse of existing solutions. Due to the strong cost pressure in the aerospace market, industry is forced to optimise their lifecycle processes and tool environments.

In order to address these challenges, we propose a Lean Systems Engineering (LSE) approach. *Lean Thinking* is a methodology which aims to deliver value to the customer while cutting out waste inadvertently generated by the process, see **Figure 1**.

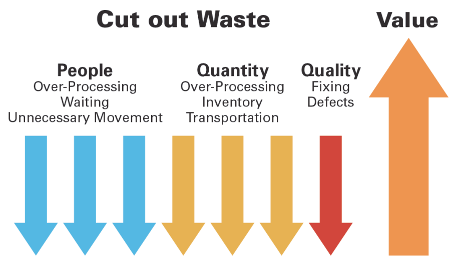


Figure 1. Lean thinking [17].

The lean methodology can be traced back to the success of the Toyota Production System (TPS) and Taiichi Ohno who is considered as the father of the TPS. J.P. Womack and D.T. Jones [18] published the basic principles in their book on Lean Thinking:

- Value,
- Value streams,
- Flow,
- Pull,
- Perfection.

In order to advance lean thinking in systems engineering, INCOSE has established the LSE Working Group. One significant result of this working group is a list of 194 practices and recommendations of systems engineering based on lean thinking, the so-called lean enablers [1]. In this paper we refer to the following lean enablers:

Map the value stream:

- Map the systems engineering value stream: have *cross-functional* stakeholders work together to build the agreed value stream.
- Plan for frontloading: anticipate and plan to resolve as many downstream issues and risks as early as possible to prevent downstream problems. Plan early for consistent robustness and "*first time right*".
- Plan to develop only what needs developing: *promote reuse* and sharing of program assets.
- Plan leading indicators and metrics to manage the program: *use metrics* structured to motivate the right behaviour.

Make value flow continuously along the value stream:

- Clarify, derive, prioritize requirements early: *use architectural methods and modelling* for system representations (prototypes, models, simulations) that allow interactions with customers.
- Promote smooth systems engineering flow: *minimize handoffs* to avoid rework.
- Make program progress visible to all: *make work progress visible* and easy to understand to all, including external customer.
- Use lean tools: use lean tools to *promote the flow of information* and minimize handoffs.

Pursue perfection:

- Strive for excellence of systems engineering processes: build in robust quality at each step of the process, and resolve and do not pass along problems. *Apply basic PDCA (Plan-Do-Check-Act) method* to problem solving.
- Develop perfect communication, coordination and collaboration policy across people and processes: *ensure timely and efficient access to centralized data*.

In the remainder of this chapter we will introduce four aims of the proposed lean systems engineering approach that allow to cover the lean enablers mentioned above.

Our first aim is to significantly *improve the quality of system specifications*, since reducing defects in requirements is the earliest opportunity in the lifecycle to save money [8]. This is accomplished by the following intertwined activities:

- Perform linguistic analysis of requirements supported by domain ontologies to obtain well-formed requirements which fulfil given quality characteristics.
- Conduct system modelling guided by model-based systems engineering (MBSE) best practice to ensure consistency and completeness of the requirements.
- Define requirements-based test cases to guarantee verifiability of requirements.
- Perform simulations of the system model and execute model-based tests to conclude the validation of the system specification.

Our second aim is to *standardize the development of variants* that share commonalities and *promote reuse*. In past projects substantial effort was spent for the development of each single variant. Thus, we introduce product family engineering practices leveraging reuse, allowing shorter time-to-market, reducing development cost, while taking into account the variability and diversity of users and customers [2].

Our third aim is to *integrate model-based system engineering with functional safety analysis* and *foster cross-functional work*. Thus, safety aspects are considered early in the system development process. Furthermore, the approach allows safety and system engineers sharing common artefacts. As a consequence, there is substantial potential for cost savings and quality improvements [14].

Our fourth aim is to *establish a tightly integrated systems engineering environment (SEE)* with interoperable tools providing requirements management, requirements quality analysis, modelling, safety analysis, simulation and test capabilities. This SEE shall support end-to-end traceability across different tools, enable efficient access to lifecycle data, support team collaboration and communication, and provide visual dashboards for process measurements. The tool integration relies on the Linked Data approach [15] and leverages interoperability standards such as Open Services for Lifecycle Collaboration (OSLC) [16].

In this paper, we present a demonstrator based on an open platform for lifecycle tool integration and an experimental case study which was performed by Airbus Defence and Space in the frame of the ARTEMIS Joint Undertaking research project CRYSTAL[1] in order to validate the approach from an industrial point of view.

The remainder of the paper is organized as follows. Section 2 introduces the industrial case study and the envisioned SEE. Sections 3 to 9 exemplify the approach. Section 10 concludes the paper and provides an outlook to future work.

2. INDUSTRIAL CASE STUDY

Airbus Defence and Space develops avionic systems that support helicopter pilots in degraded visual environments (DVE) which can be caused by e.g. rain, fog, sand and snow. Many accidents can directly be attributed to such DVE where pilots often loose spatial and environmental orientation (see **Figure 2** on the left side). In this case study we employ the landing symbology function which is part of the "Pilot Assistance Landing" capabilities of the situational awareness suite Sferion™. Other Sferion™ capabilities are "Pilot Assistance In-flight" or "Obstacle Warning".

The landing symbology function supports helicopter pilots during the landing approach. It enables the pilot to mark the intended landing position on ground using a head-tracked HMS/D (Helmet Mounted Sight and Display) and HOCAS (Hands on Collective and Stick). During the final landing approach the landing symbology function enhances the spatial awareness of flying crews by displaying 3D conformal visual cues on the HMS/D (see **Figure 2** on the right side). Additionally, obstacles residing in the landing zone can be detected.

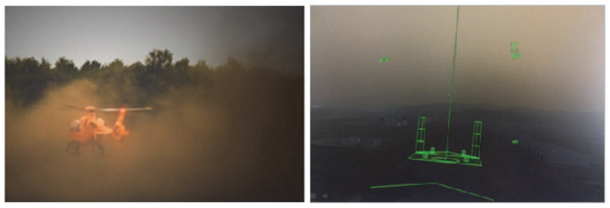


Figure 2. Landing aid in degraded visual environments.

[1] Critical System Engineering Acceleration, http://www.crystal-artemis.eu/

The situational awareness suite Sferion™ constitutes a product family to be deployed on different helicopter platforms. The reuse of development assets, e.g. requirements, design or test cases, can be significantly improved by the application of product family engineering practices. In this case study we applied the product family engineering practice "Scoping". The practices "Domain Requirements Engineering" and "Domain Design" have been applied on system level to develop system requirements and the system design for the product family [2].

The envisaged SEE includes tools and databases to support Application Life Cycle Management (ALM) and Product Lifecycle Management (PLM), see **Figure 3**. For interoperability, each tool has to provide a connector that is based on open interoperability standards (IOS) such as OSLC[2]. The communication between the tools (e.g. sending of requests to other tools, receiving data from tools) is realised by well-established web protocols.

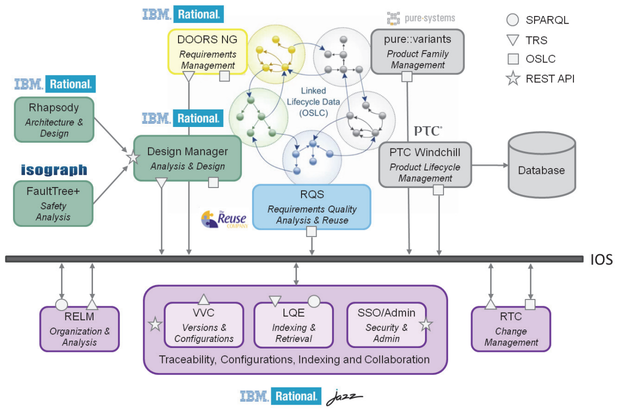


Figure 3. Envisioned seamlessly integrated SEE.

The following sections summarize the steps performed in the case study and describe the related tool chain:

- Define product family scope.
- Develop system requirements.
- Analyse and improve requirements quality.
- Develop system model.
- Perform safety analysis.
- Develop and perform model-based tests.
- Create product variants.

3. PRODUCT FAMILY SCOPING

First, the characteristics of potential products (e.g. low-cost or high-end variants) are categorized. In the Sferion™ product family three main configurations have been identified: class 2, 3 and 4.

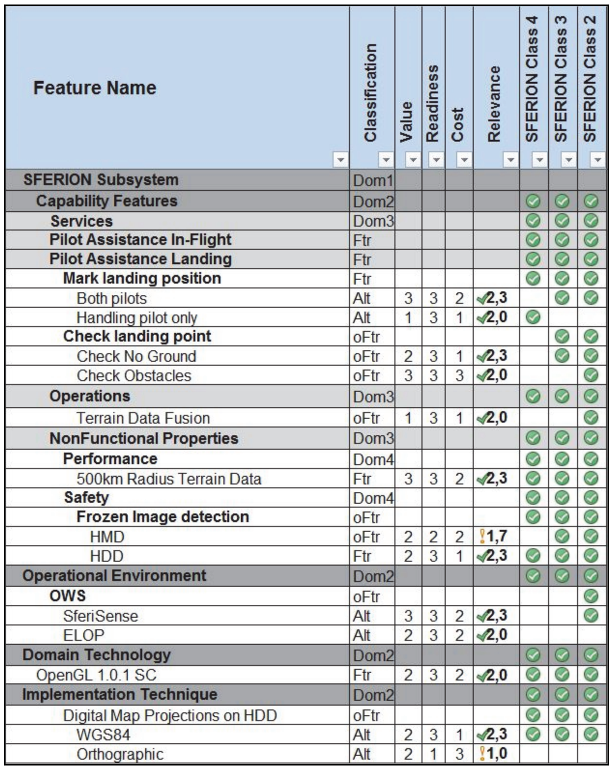
[2] Open Services for Lifecycle Collaboration, http://open-services.net /

Then, features of the product family are identified by [7]:

- Analysing the capability in terms of end user visible services, internal operations needed to provide the end user visible services and definition of non-functional properties, e.g. performance.
- Analysing the operational environment of the product family to define the context, e.g. external systems and interfaces.
- Analysing domain technology required to implement services or operations, e.g. development approaches or tools.
- Analysing implementation technique to indicate key design and implementation decisions used to implement other features.

Features are refined in terms of commonality and variability rather than describing all details. Common features are described on top level. Feature variations are refined until no variation is exposed among products. An initial classification of each feature is performed (e.g. feature is optional; feature is refined into a range of alternatives), see **Figure 4**. Then, each feature is allocated to the intended product variant.

Finally, each feature is assessed by the stakeholders with respect to customer value, development risk and cost yielding a relevance indicator. A threshold for the relevance is defined in order to get the list of features which should be in the scope of the product family. The results of the scoping process are recorded in an Excel sheet – the Product Feature Matrix (PFM).



Feature Name	Classification	Value	Readiness	Cost	Relevance	SFERION Class 4	SFERION Class 3	SFERION Class 2
SFERION Subsystem	Dom1							
Capability Features	Dom2					✓	✓	✓
Services	Dom3					✓	✓	✓
Pilot Assistance In-Flight	Ftr					✓	✓	✓
Pilot Assistance Landing	Ftr					✓	✓	✓
Mark landing position	Ftr					✓	✓	✓
Both pilots	Alt	3	3	2	✓2,3		✓	✓
Handling pilot only	Alt	1	3	1	✓2,0	✓		
Check landing point	oFtr						✓	✓
Check No Ground	oFtr	2	3	1	✓2,3		✓	✓
Check Obstacles	oFtr	3	3	3	✓2,0			✓
Operations	Dom3					✓	✓	✓
Terrain Data Fusion	oFtr	1	3	1	✓2,0			✓
NonFunctional Properties	Dom3					✓	✓	✓
Performance	Dom4					✓	✓	✓
500km Radius Terrain Data	Ftr	3	3	2	✓2,3	✓	✓	✓
Safety	Dom4					✓	✓	✓
Frozen Image detection	oFtr					✓	✓	✓
HMD	oFtr	2	2	2	!1,7		✓	✓
HDD	Ftr	2	3	1	✓2,3	✓	✓	✓
Operational Environment	Dom2					✓	✓	✓
OWS	oFtr							✓
SferiSense	Alt	3	3	2	✓2,3			✓
ELOP	Alt	2	3	2	✓2,0			
Domain Technology	Dom2					✓	✓	✓
OpenGL 1.0.1 SC	Ftr	2	3	2	✓2,0	✓	✓	✓
Implementation Technique	Dom2					✓	✓	✓
Digital Map Projections on HDD	oFtr					✓	✓	✓
WGS84	Alt	2	3	1	✓2,3	✓	✓	✓
Orthographic	Alt	2	1	3	!1,0			

Figure 4. Extract of a Product Feature Matrix (PFM).

Important descriptive goals of the feature model are:

- Understanding of the variability among products.
- Communication of requirements in terms of features.
- Specification of product variants.

Additionally, prescriptive goals of the feature model are important for this case study:

- Guidance for the development or transformation of core assets which are in our case study the product family system requirements and product family architecture.
- Guidance for variant derivation of the system requirements and the system architecture model.

The variability modelling tool pure::variants was selected to support the prescriptive feature modelling goals. Based on the PFM, a variability model has been set up with pure::variants in order to formalize the results of the product family scoping in a feature tree, see **Figure 5**.

Subsequently, features, commonalities, variabilities and structural relationships have been added to the variability model.

The feature tree was manually optimized to represent variabilities and commonalities rather than functional dependencies like a function call hierarchy.

For example, the mandatory feature "Mark landing position" can be realized in two ways: either the handling pilot only or both pilots are allowed to set the landing position. All product variants provide the safety feature "FrozenImageDetection". However, the frozen image detection on "HMD" is optional.

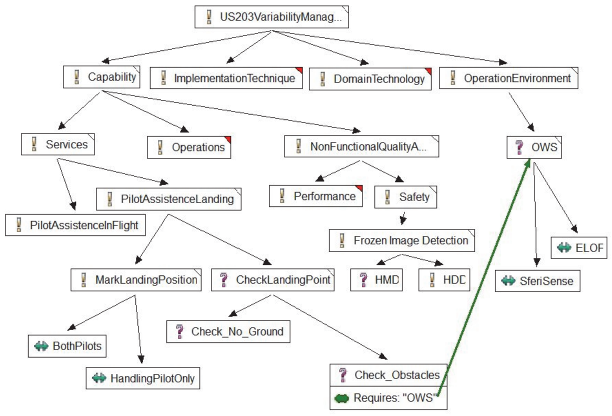


Figure 5. Feature model describing the variabilities.

The main advantage of the feature modelling approach is that based on the feature model, validation mechanisms assure that only valid product configurations are defined.

In order to provide guidance for variant derivation it is important to add composition rules to optional and alternative features. Essential composition rules are:

- Mutual dependency relationship: features that shall be selected along with the designated one.
- Mutual exclusion relationship: features that shall not be selected along with the designated one.

For example, if "Check_Obstacles" is selected, an Obstacle Warning System (OWS) is required. In this case, one of the sensor equipments "ELOP" or "SferiSense" has to be selected.

4. REQUIREMENTS DEVELOPMENT

After the definition of the product family scope, a first set of system requirements covering all features of the product family and derived from stakeholder needs is defined using the requirements management tool DOORS NG. The requirement type is assigned. This enables filtering requirements according to their type, e.g. gathering all functional requirements as input for the subsequent functional analysis.

In order to support the requirement authoring activity, we use two additional tools from the Requirements Quality Suite (RQS) [11]:

- Knowledge Manager (kM), which is used to create and maintain the domain ontology and requirements patterns.
- Requirements Authoring Tool (RAT), which provides on-the-fly guidance for the requirements specification activity [9].

First, we developed a domain ontology by defining terms, abbreviations, agreed-upon concepts and relations that hold in our application domain. Then, we defined requirements patterns by defining sequences of fixed syntax elements for each type of pattern to cover the relevant requirements statements [10]. Requirements patterns can be specified in an elementary way and then concatenated to cover more complex sentence structures, allowing keeping the number of required patterns low while at the same time having a high flexibility. The majority of the functional requirements of our case study could be covered with a few set of patterns.

Figure 6 shows how a Natural Language (NL) requirement is represented by a pattern. The pattern consists of fixed syntactic elements and variable elements. The latter are mapped to semantic concepts in the domain ontology such that the requirement can formalized by a semantic graph.

Using requirement patterns is an effective way to mitigate many types of ambiguity in NL requirements and to enable advanced formal analysis of these requirements, see Section 5. For instance, we can detect requirements formulated in passive voice or any requirement structure that is not well-formed like "It shall be possible..." which lacks an actor.

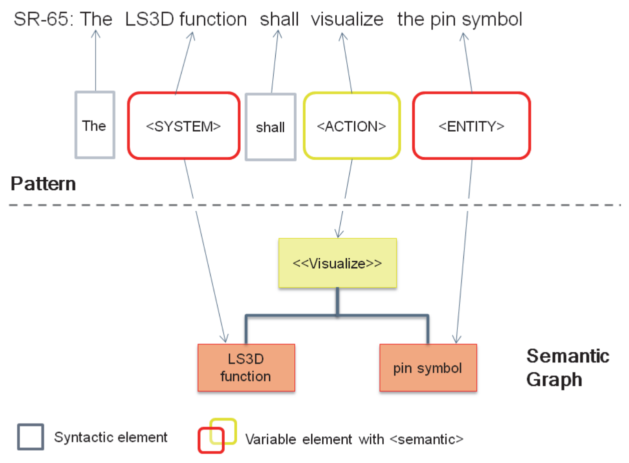


Figure 6. Semantic analysis.

The RAT tool uses the concepts of the domain ontology together with the set of pre-defined requirement patterns managed in the common asset repository to provide a list of suggestions the requirements author can directly build on when defining textual requirements. If done manually, pattern conformance checking can be cumbersome, particularly when requirements change frequently. The tool gives real-time feedback whether a selected pattern is matched or not and also on the quality. Additionally, terms of the controlled vocabulary fitting with the next matched pattern element are suggested, which force the authors to use consistently agreed-upon domain terms.

Having a proactive and interactive guidance that tries to improve requirements quality while actually writing requirements, is a real benefit. It is essential that errors in requirements are found early to avoid cost and error propagation later in the project, as well as reducing rework and modification loops. According to Boehm's law [6], revealing a defect in the requirements stage is 100 times cheaper than fixing one in coding. Additionally, using a domain ontology allows a better know-how transfer between domain experts and requirements authors in order to achieve a common understanding on the set of requirements between all project stakeholders. Dealing with all these new tools beside DOORS NG as requirements management tool bring new challenges and a systematic process for ontology creation and maintenance is needed as well as a new engineering role: the knowledge manager.

In order to integrate the set of system requirements with the variability model, formal relationships between individual system requirements and features need to be established. The OSLC connector of pure::variants reads all requirements including the allocated features from DOORS NG and initializes the family model defining the reusable artefacts accordingly. In case of changes, the family model can be synchronized again with changes performed in DOORS NG.

5. REQUIREMENTS QUALITY ANALYSIS

The quality of requirements has a major impact on project success. Badly written requirements are a well-known source of project failure. Moreover, the quality of requirements should be secured before reusing them in different product variants. For this reason, quality metrics are employed to measure the requirements quality and to identify language defects, see **Figure 7**. Some quality metrics are: readability, use of passive voice, ambiguous terms, negations, abuse of connectors, undefined acronyms, and inconsistent use of measurement units. The INCOSE Requirements Working Group [4] and ISO/IEC 29148 [5] provide an exhaustive definition and justification of requirements quality characteristics.

Quality analysis can be done on different levels as depicted in **Figure 7**:

- Textual analysis: e.g. size or readability.
- Lexical and syntactic analysis: e.g. ambiguous terms and passive voice.
- Semantic analysis based on domain ontologies: e.g. overlapping requirements and completeness.

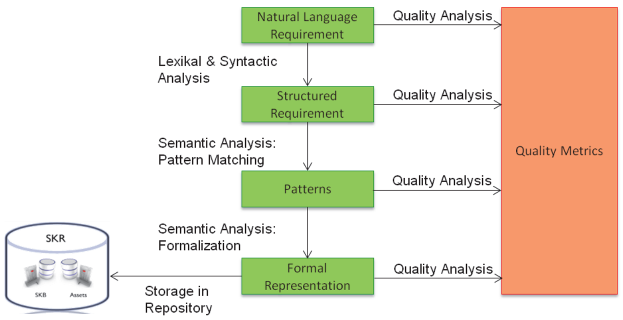
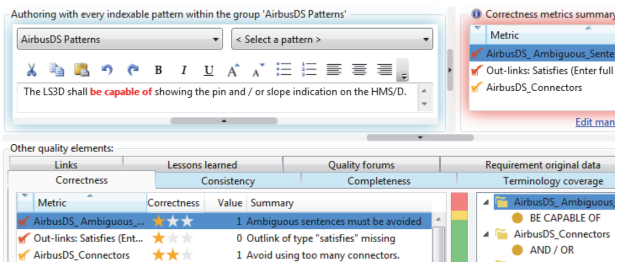


Figure 7. Derivation of requirement quality metrics.

We use the Requirements Quality Analyser (RQA) for DOORS NG that provides an automatic quality evaluation of NL requirements by performing lexical and linguistic checks as well as semantic analysis based on the developed domain ontology and requirement patterns, see Section 4.

RQA comprises more than 60 pre-defined metrics, from which a subset may be selected for quality analysis. Every single requirement is analysed one by one and a series of indicators (e.g. readability, ambiguous phrases, and traceability) is determined for every requirement. Every indicator is then transformed into a qualitative value by the associated quality function. During the evaluation, every quality metric rated as medium or low will generate some hints and suggestions for improvement. **Figure 8** depicts measurement results for one requirement. Most relevant findings are:

- Ambiguous sentences: “be capable of”
- Ambiguous use of connectors: “and/or”
- Missing traceability: outlink of type “satisfies” is missing



Authoring with every indexable pattern within the group 'AirbusDS Patterns'

AirbusDS Patterns | < Select a pattern >

The LS3D shall **be capable of** showing the pin and / or slope indication on the HMS/D.

Correctness metrics summar

Metric
AirbusDS_ Ambiguous_Sente
Out-links: Satisfies (Enter full
AirbusDS_Connectors

Edit man

Other quality elements:

Links | Lessons learned | Quality forums | Requirement original data

Correctness | Consistency | Completeness | Terminology coverage

Metric	Correctness	Value	Summary
AirbusDS_ Ambiguous_...	★★★	1	Ambiguous sentences must be avoided
Out-links: Satisfies (Ent...	★☆☆	0	Outlink of type "satisfies" missing
AirbusDS_Connectors	★★☆	1	Avoid using too many connectors.

AirbusDS_ Ambiguous
- BE CAPABLE OF

AirbusDS_Connectors
- AND / OR

Figure 8. RQA quality report.

There is also another view for the whole set of requirements in a DOORS NG module, which provides a quick overview of the requirement quality in this module. Moreover, the most frequent quality issues are displayed which allows identifying focused corrective actions.

In addition, there exist several ontology-based metrics which give feedback whether concepts are properly used at the right level of abstraction. For instance we can make use of metrics related to the Product Breakdown Structure (PBS), which is represented within the ontology. In lower level documents, we activate the detection of "compound" terms to force the authors to specify the concrete sub-elements as defined in PBS. On the other hand for high-level documents, we can check whether "part" terms have been used to avoid that there exist too specific terms there.

Some additional metrics make use of the categorization of terms within a specific semantic cluster. Terms which represent a system, component or a sub-component of the PBS are assigned to e.g. the semantic cluster "SYSTEM". These clusters are used within the pattern definition to restrict the allowed terms within the defined patterns.

Additionally, we make use of the common asset repository for advanced requirements analysis like consistency and completeness of the whole set of requirements. When dealing with complex systems and a high number of requirements as well as many hierarchal levels, consistency and completeness checks are difficult manual tasks. Here are some examples of automated quality checks:

- Identification of inconsistent use of measurements units.
- Identification of redundant requirements by means of their respective formalization as a semantic graph. Overlapping requirements between different hierarchical levels may be an indication for an insufficient refinement.
- Completeness assessment whether any requirements of a specific category (e.g. safety, performance…) are possibly missing based on the matched pattern groups within the ontology.

Finally, analysis results are interpreted; requirements are improved and updated in DOORS NG. In addition, the domain ontology needs to be updated and validated by changing concepts and pattern formalizations to improve guidance and analysis results.

The whole set of RQS tools provides an effective support in defining well-structured requirements and performing automated analysis of requirements. With this approach, the review is more efficient since the amount of manual work is reduced; all trivial checks are performed by the tool and can even be corrected by the author itself. The reviewer can then concentrate on "difficult" points, the assessment results are reproducible and do not depend on individual reviewers' subjective opinion. Quality criteria are clear right from the beginning and hints are provided which motivate authors and drive the quality improvement process of requirements according to the PDCA (Plan-Do-Check-Act) cycle [8].

6. SYSTEM MODEL DEVELOPMENT

In order to support early validation of the system specification and improve communication within the project team and with stakeholders, we adopt a Model-based Systems Engineering approach (MBSE), which constitutes three different viewpoints:

- Functional viewpoint
- Logical viewpoint
- Physical viewpoint

The aim of the *functional viewpoint* is to describe in detail from a technical perspective the functions and behaviour the intended system shall provide, the interaction via identified interfaces with external systems, users and operators, and the interaction and dependencies between the different functions (black box approach). The functional analysis is performed with Rhapsody based on the MBSE methodology Harmony/SE [3] and results in use case diagrams, activity diagrams, sequence diagrams, block diagrams and statechart diagrams.

First, the system requirements are accessed from DOORS NG and displayed in Rhapsody. The goal is to prove the complete consideration of all relevant system requirements during the functional analysis process. Then, the system context with interfacing actors is defined with a use case diagram. Other use case diagrams may be created to group the functional scope accordingly. Subsequently, the different functional flows are captured using activity diagrams. In the next step, a set of sequence diagrams consistent with the activity diagram and describing the behaviour of the system in a particular situation, is derived. Operations are derived from actions. Events and related event receptions are defined in relation with external actors. The identified events are linked to data items that are transported across interfaces. With the identified sequence diagrams, the system ports and interfaces including all in- and outgoing events can be defined using an internal block diagram. To complete the functional analysis, a statechart is added to the system block to describe its state-based behaviour.

Thanks to the formal semantics of statecharts, code can be generated that implements the state-based behaviour of the system. Although the generation of production code is possible, we only use code generation for the execution of the functional model since we do not intend to qualify the code generator. When executing the model, the system behaviour can visually be inspected with the help of animated statecharts. Model execution is a powerful method to check the correctness and completeness of the functional model and the related system requirements. For this purpose simulations can be setup either by manually stimulating the system with events or by implementing model-based tests. The latter is described in more detail in Section 8.

The aim of the *logical viewpoint* is to identify an architecture of logical blocks which realises the functions and behaviour identified during the functional analysis (white box approach). In this process also non-functional aspects are considered, e.g. reduction of the number of interfaces, segregation of functions with different criticalities (see Section 7), reuse of available functions (see Section 9), integration of external suppliers, and utilization of COTS elements. The system block representing the complete functional scope is decomposed into logical blocks. Activity diagrams and sequence diagrams are refined considering partitions and lifelines related to logical blocks. System internal ports and interfaces are added for connecting the logical blocks. The state-based behaviour of all logical blocks is defined which allows to validate the logical architecture using model execution.

The aim of the *physical viewpoint* is to find a suitable system architecture consisting of system elements (e.g. hardware or software items) that allows to implement the logical blocks identified in the logical viewpoint, while considering the business needs and non-functional constraints. In order to derive a candidate system architecture, the logical blocks of the logical architecture are mapped to physical elements of the physical architecture. In the same way, the logical interfaces are mapped to physical interfaces. Both physical elements and physical interfaces need to comply with associated performance requirements. Additionally, the properties of the selected physical elements (e.g. power consumption, memory size, …) are identified and modelled. Based on a set of prioritized criteria, different candidate solutions can be assessed during a trade-off study. Finally, a solution is jointly selected by all stakeholders.

Moreover, traceability is established between the system requirements in DOORS NG and the model elements in Rhapsody for all viewpoints of the system model to support coverage and impact analysis. For example, when requirements are changed, a suspect indicator is automatically added to the trace which helps maintaining consistency.

7. SAFETY ANALYSIS

Based on system requirements and the functional analysis of the system, hazards can be identified. Contributing system functions, as they have been identified through the functional analysis in the black box system model, are classified according to their related criticality level. The classification of the system functions as part of the Functional Hazard Assessment (FHA) considers failure modes (e.g. total loss, partial loss, inadvertent provision, and erroneous provision) of functions and the flight phase (e.g. takeoff, approach, and landing) where this failure conditions occur. Further results of the hazard assessment are derived requirements. These new safety requirements are then considered in the white box system model within the logical and physical architecture, where the system architecture has to show compliance not only to the initial functional requirements but also to all safety related requirements.

Figure 9 illustrates the black box approach that comprises the functional analysis and shows how the FHA process is integrated with the functional viewpoint.

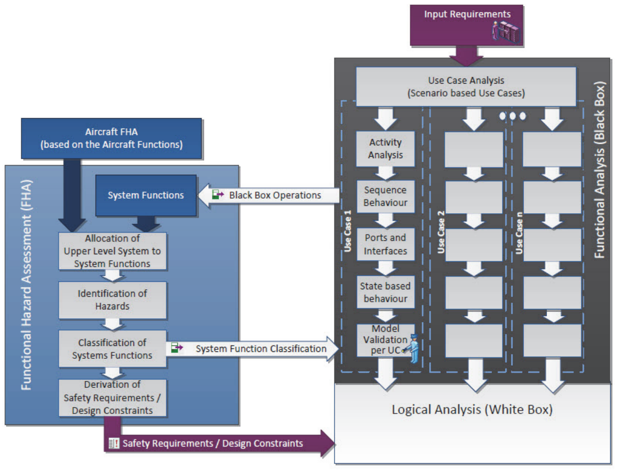

Input Requirements
Use Case Analysis (Scenario based Use Cases)
Aircraft FHA (based on the Aircraft Functions)
Functional Hazard Assessment (FHA)
System Functions
Black Box Operations
Allocation of Upper Level System to System Functions
Identification of Hazards
Classification of Systems Functions
System Function Classification
Derivation of Safety Requirements / Design Constraints
Safety Requirements / Design Constraints
Use Case 1
Use Case 2
Use Case n
Activity Analysis
Sequence Behaviour
Ports and Interfaces
State based behaviour
Model Validation per UC
Functional Analysis (Black Box)
Logical Analysis (White Box)

Figure 9. Black box approach.

The white box approach consists of the logical viewpoint, in which an architecture is iteratively developed preventing all failures which have been identified through the FHA. Since also the physical viewpoint, that involves the identification of real interfaces and hardware components, is emerging from the white box approach, fault trees can be elaborated with a failure condition as a starting point and refined up to physical elements, where acceptable probabilities for the occurrence of failure conditions are assigned.

Figure 10 shows the white box approach comprising the logical and the physical viewpoints.

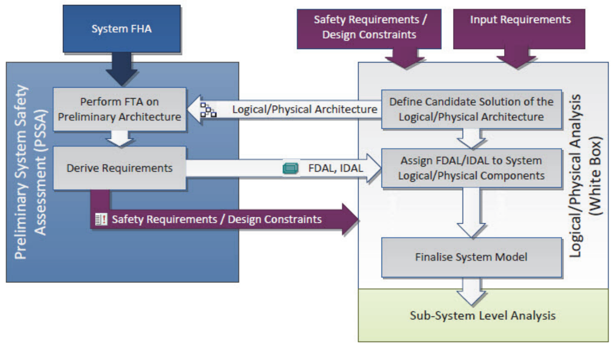

System FHA
Safety Requirements / Design Constraints
Input Requirements
Preliminary System Safety Assessment (PSSA)
Perform FTA on Preliminary Architecture
Logical/Physical Architecture
Define Candidate Solution of the Logical/Physical Architecture
Derive Requirements
FDAL, IDAL
Assign FDAL/IDAL to System Logical/Physical Components
Safety Requirements / Design Constraints
Finalise System Model
Logical/Physical Analysis (White Box)
Sub-System Level Analysis

Figure 10. White box approach.

System modelling including functional, logical and physical viewpoints is performed with Rhapsody, whereas safety requirements and failure condition lists are kept in DOORS NG. Fault trees as part of the Preliminary System Safety

Assessment (PSSA) are generated with isograph FaultTree+. As shown in **Figure 11**, system functions identified from black box analysis are made available for the FHA elaborated in DOORS NG. As a result from the FHA, safety requirements and design constraints are available for the white box analysis in Rhapsody. Moreover, they could have an impact on the black box analysis since new functions may have to be elaborated to mitigate the safety concerns. Using the FHA output and the white box analysis refinement into components, a PSSA is developed including fault trees. Resulting from these fault trees FDALs (Function Development Assurance Level) and IDALs (Item Development Assurance Level) are then allocated to components. This proceeding allows establishing traceability between safety requirements, function classifications and the system model.

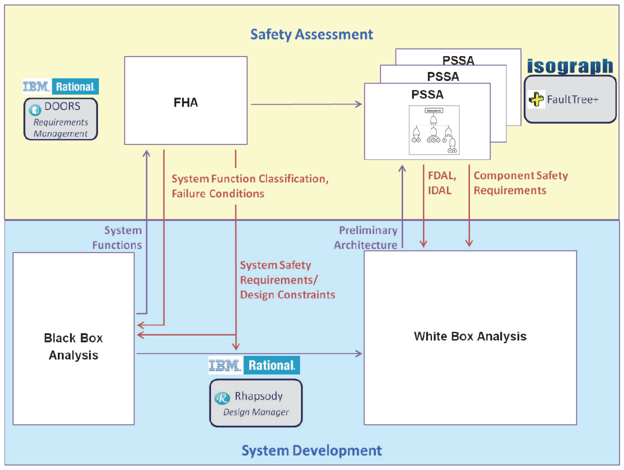


Figure 11. Tooling and exchanged artefacts.

8. MODEL-BASED TESTING

The notion of Model-based Testing (MBT) refers to the application of models for automation of testing activities as well as modelling of test artefacts. In the following MBT is used to automatically generate test artefacts from the black box system model which represents the system under test (SUT). As discussed in Section 6 this model allows the simulation of the system behaviour. The intention is to verify that this simulation is compliant with the expected behaviour as defined by the system requirements [12].

The underlying process can be divided into the following phases [13]:

1. Modelling of the SUT and/or its environment. Creation of the test architecture.
2. Generation of executable test cases from the black box system model.
3. Test execution on the SUT and assignment of verdicts.
4. Analysis of the test results.

Figure 12 provides a graphical overview of the MBT process.

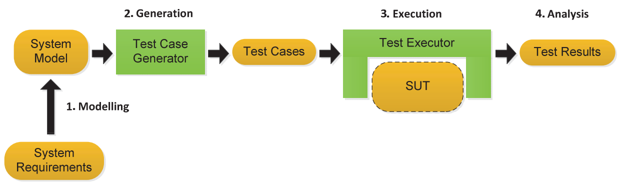


Figure 12. MBT process.

Modelling of the behaviour of the SUT is already described in detail in Section 6. TestConductor (TC), the test execution and verification engine of Rhapsody is employed for creating the test architecture of the selected SUT. Test architectures comprise all artefacts which are needed for test automation, e.g. test components are created for stubbing of external interfaces.

For automatic generation of executable test cases the Rhapsody Automatic Test Generation add-on (ATG) is applied. By analysis of the specified system model ATG automatically generates suites of test cases which are specified by UML sequence diagrams as depicted in **Figure 13**. Test stimuli are generated which allow observing the behaviour of the SUT. In this format the test cases are ready for execution.

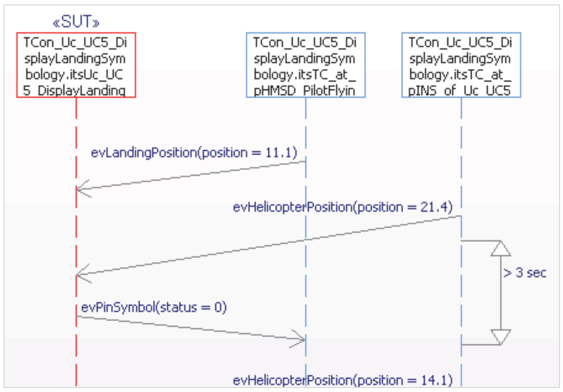
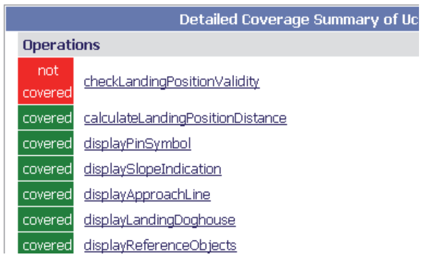

«SUT»
TCon_Uc_UC5_DisplayLandingSymbology.itsUc_UC5_DisplayLanding
TCon_Uc_UC5_DisplayLandingSymbology.itsTC_at_pHMSD_PilotFlyin
TCon_Uc_UC5_DisplayLandingSymbology.itsTC_at_pINS_of_Uc_UC5
evLandingPosition(position = 11.1)
evHelicopterPosition(position = 21.4)
> 3 sec
evPinSymbol(status = 0)
evHelicopterPosition(position = 14.1)

Figure 13. Test case generated by ATG.

TC is employed for executing the test cases. Test verdicts are provided which for example allow to identify all requirements which are not satisfied by the black box model or state to which extent the system model has been covered by the executed tests.

After completion of the test execution a detailed report summarizing all relevant information including test results and model coverage analysis is generated, see **Figure 14**.



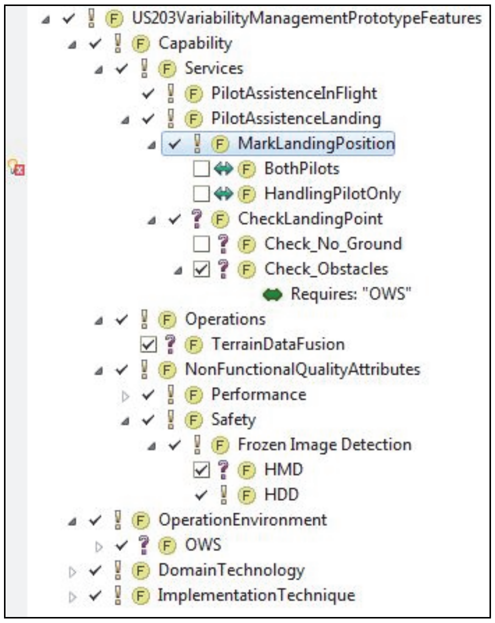
Detailed Coverage Summary of Uc

Operations	
not covered	checkLandingPositionValidity
covered	calculateLandingPositionDistance
covered	displayPinSymbol
covered	displaySlopeIndication
covered	displayApproachLine
covered	displayLandingDoghouse
covered	displayReferenceObjects

Figure 14. Model coverage report generated by TC.

9. VARIANT MANAGEMENT

The definition of product variants is supported by an auto resolver based on the feature model (see Section 3). **Figure 15** illustrates how the configuration process is performed. Mandatory features like "MarkLandingPosition" are automatically selected. Optional features without a composition rule like "TerrainDataFusion" need to be selected manually. Features affected by composition rules are selected or deselected by the auto resolver, e.g. if "Check_Obstacles" is user-selected, the feature "OWS" is automatically selected by the auto resolver.

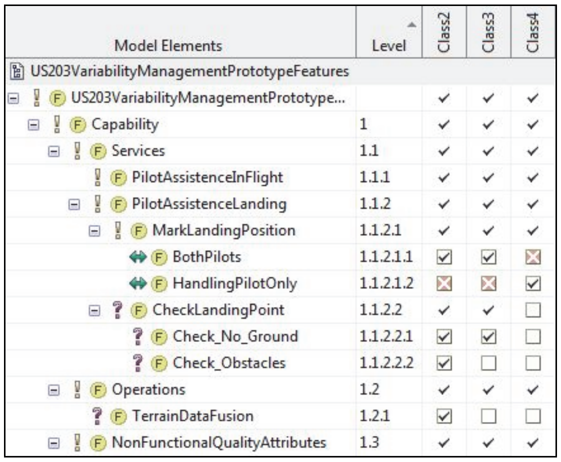


```
◢ ✓ ! F US203VariabilityManagementPrototypeFeatures
   ◢ ✓ ! F Capability
      ◢ ✓ ! F Services
           ✓ ! F PilotAssistenceInFlight
         ◢ ✓ ! F PilotAssistenceLanding
            ◢ ✓ ! F MarkLandingPosition
                 ☐ ⇔ F BothPilots
                 ☐ ⇔ F HandlingPilotOnly
            ◢ ✓ ? F CheckLandingPoint
                 ☐ ? F Check_No_Ground
               ◢ ☑ ? F Check_Obstacles
                      Requires: "OWS"
      ◢ ✓ ! F Operations
           ☑ ? F TerrainDataFusion
      ◢ ✓ ! F NonFunctionalQualityAttributes
         ▷ ✓ ! F Performance
         ◢ ✓ ! F Safety
            ◢ ✓ ! F Frozen Image Detection
                 ☑ ? F HMD
                 ✓ ! F HDD
   ◢ ✓ ! F OperationEnvironment
      ▷ ✓ ? F OWS
   ▷ ✓ ! F DomainTechnology
   ▷ ✓ ! F ImplementationTechnique
```

Figure 15. Configuration of a product variant.

Inconsistencies in the feature model or missing feature decisions, e.g. in case of alternative feature selections, are highlighted in order to conclude a valid variant description, see missing decision on the alternatives of feature "MarkLandingPosition" in **Figure 15**.

Product variants are compared and validated using the product variant matrix view, see **Figure 16**.



Model Elements	Level	Class2	Class3	Class4
US203VariabilityManagementPrototypeFeatures				
US203VariabilityManagementPrototype...		✓	✓	✓
Capability	1	✓	✓	✓
Services	1.1	✓	✓	✓
PilotAssistenceInFlight	1.1.1	✓	✓	✓
PilotAssistenceLanding	1.1.2	✓	✓	✓
MarkLandingPosition	1.1.2.1	✓	✓	✓
BothPilots	1.1.2.1.1	☑	☑	☒
HandlingPilotOnly	1.1.2.1.2	☒	☒	☑
CheckLandingPoint	1.1.2.2	✓	✓	☐
Check_No_Ground	1.1.2.2.1	☑	☑	☐
Check_Obstacles	1.1.2.2.2	☑	☐	☐
Operations	1.2	✓	✓	✓
TerrainDataFusion	1.2.1	☑	☐	☐
NonFunctionalQualityAttributes	1.3	✓	✓	✓

Figure 16. Product matrix view.

Each variant configuration is input to the transformation process that creates system requirements, system models, and tests for product variants. Finally, system specifications can be generated automatically capturing the information gathered during the requirements and system model development in a structured way. The generated documents can be used to perform formal reviews, fulfil contractual obligations or show regulatory compliance.

10. CONCLUSION

We reported on our industrial case study in the aerospace domain which was conducted in the frame of the ARTEMIS Joint Undertaking project CRYSTAL. The case study and the demonstrator cover the identification of variabilities, the construction of a feature model using pure::variants, the development and quality analysis of system requirements using DOORS NG and RQS, the elaboration of a system model using Rhapsody and safety analysis tightly integrated within the MBSE process. Moreover, requirements-based test cases are developed with TestConductor and ATG that allow verifying the system model by simulation runs.

Applying our approach to the industrial case study has produced promising results and addresses the following lean enablers (see Section 1): the *basic PDCA method is applied* during the requirements quality analysis to resolve problems related with badly written requirements. The *use of architectural methods and models* in the model-based systems engineering approach allows to clarify requirements early. Through the combination of linguistic analysis, model-based systems engineering, and model-based testing the quality of system specifications can significantly be improved. This contributes to the "*first time right*" objective. The tightly integrated safety analysis supports *cross-functional* team work between systems and safety engineers, *promotes the flow of information* and *minimizes handoffs* and rework. Additionally, the product family engineering approach allows to derive system specifications for different variants and *promotes reuse* of shared artefacts. The integrated SEE supports *efficient access to systems engineering data*. It enables to *make the work process visible* using metrics and facilitates end-to-end traceability analysis across different tools.

The experience gained from the realisation of the case study can be summarized as follows:

- Setting up useful ontologies that effectively support authoring and quality analysis of requirements still requires a lot of experience and specific skills. Therefore, the depth and thoroughness of quality analysis has to be carefully planned depending on the objectives and boundary conditions of each project.
- In the case study a monolithic feature model was chosen. In more complex systems a modular feature model structure might be considered. A modular feature model approach is easier to understand and provides a better separation of concerns but has the burden of maintaining the

traceability and dependencies between the feature models.

- The tool integration approach of the SEE allows loosely coupled tools to share and link data based on standardized and open web technologies. It should be noted that an increasing number of tool vendors support the OSLC standard. However, standardisation needs to be extended in order to support domains not covered yet, e.g. variability management and safety analysis.

We are currently working to further develop the integration of safety analysis into the MBSE approach. In addition, we are using formal specifications that can be analysed with model checking techniques. Moreover, we are adopting a more holistic approach of configuration management which provides means to define global system configurations containing all relevant types of artefacts (e.g. requirements, system model and test cases) including the traceability links between them.

Our future work will advance and improve component-based and model-based systems engineering practices. Components will comprise executable specifications and ease virtual integration and prototyping. They will provide variation points in order to support product families by derivation of variants. Contractual specification of components enable more efficient system verification and continuous checks of functional safety properties, e.g. by formal checks performed during virtual integration. Components already being qualified provide certification evidence to support incremental qualification and certification approaches. New systems can then be constructed with components according to domain-specific reference architectures. Domain-specific languages will further ease the specification, architectural design and verification of complex systems.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no. 332830 (ARTEMIS project CRYSTAL) and German BMBF.

REFERENCES

[1] Oppenheim, B.W., *Lean for Systems Engineering with Lean Enablers for Systems Engineering*. John Wiley & Sons, 2011.

[2] Pohl, K., G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag, 2005.

[3] Hoffmann, H.-P., Deskbook Release 4.1, Model-based Systems Engineering with Rational Rhapsody and Rational Harmony for Systems Engineering. IBM Corporation, 2014.

[4] Requirements Working Group, International Council on Systems Engineering (INCOSE), "Guide for Writing Requirements", INCOSE, 2012.

[5] International Standard ISO/IEC 29148: Software and systems engineering — Life cycle processes — Requirements engineering, 2011.

[6] Boehm, B.W. and Ph. N. Papaccio, "Understanding and Controlling Software Costs", *IEEE Transactions on Software Engineering* 14 (10), October 1988, pp. 1462-1477

[7] Lee, K., K. Kang, J. Lee: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, ICSR-7, 2002.

[8] Bogusch, R., "Towards Automatic Quality Evaluation of Natural-Language Requirements", In: M. Maurer, S.-O. Schulze (eds.), *Tag des Systems Engineering, Bremen 12.-14. November 2014*, Hanser, München, 2015, pp. 401-410

[9] Fuentes, J.M., A. Fraga, J. Llorens, L. Alonso, and G. Génova, Requirements Authoring: Towards the Concept of a Standard Requirement. International Council on Systems Engineering (INCOSE) Symposium 2014, Las Vegas, 2014.

[10] Fraga, A., J. Llorens, J.M. Fuentes, and L. Alonso, Knowledge-based Requirements Engineering Process: A Guided Example. To appear in *Proc. FDL Forum on Specification & Design Languages*, München, 2014.

[11] Requirements Quality Suite, The REUSE Company, http://www.reusecompany.com/requirements-quality-suite (last visited October 13, 2015).

[12] Dias, N.A., R. Subramanyan, M. Vieira, and G. Travassos, "A survey on model-based testing approaches: A systematic review", *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, Atlanta, 2007, pp. 31-36

[13] Utting, M. and B. Legeard,. *Practical Model-Based Testing: A Tools Approach*, Elsevier Science & Technology Books, 2006.

[14] Binder, I., "Towards seamless integration of functional safety and model-based systems engineering", In: M. Maurer, S.-O. Schulze (eds.), *Tag des Systems Engineering, Bremen 12.-14. November 2014*, Hanser, München, 2015, pp. 53-62

[15] Bizer, Ch., T. Heath, and T. Berners-Lee. "Linked Data – The Story so Far". *International Journal on Semantic Web and Information Systems* 5 (3), 2009, pp. 1-22

[16] OSLC. Open Services for Lifecycle Collaboration. OASIS. http://open-services.net (last visited October 13, 2015).

[17] Lean Systems Engineering Working Group, International Council of Systems Engineering (INCOSE), Lean Systems Engineering - an Introduction, INCOSE UK, 2015.

[18] Womack, J.P. and D.T. Jones. *Lean Thinking*. Simon & Schuster, New York, 1996.

Session 19

Requirement validation

Thursday 28th, 15:00 – Ariane 1

Debugging Embedded Systems Requirements with STIMULUS: an Automotive Case-Study

Bertrand Jeannet and Fabien Gaucher
ARGOSIM SA
8-10, rue de Mayencin, 38400 St-Martin d'Hères, France
{Bertrand.Jeannet|Fabien.Gaucher}@argosim.com

Abstract—**In a typical software project, 40% to 60% of design bugs are caused by faulty requirements that generate costly iterations of the development process as specifications need to be redefined, design and implementation modified accordingly, and then retested. The major reason for this situation is that no practical tool exists for debugging requirements while drafting specification, and the many tools that exist for requirement management and traceability do not address this problem.**

STIMULUS provides an innovative solution for the early debugging and validation of functional real-time systems requirements. It provides a high-level language to express textual yet formal requirements, and a solver-driven simulation engine to generate and analyze execution traces that satisfy requirements. Visualizing what systems will do enables system architects to discover ambiguous, incorrect, missing or conflicting requirements before the design begins.

We demonstrate the use of STIMULUS on the specification of automatic headlights from the automotive industry. We show how this unique simulation technique enables to discover and to fix ambiguous and conflicting requirements, resulting in a clear and executable specification that can be shared among engineers.

Keywords—Requirement Engineering, Real-time Embedded Systems, Domain Specific Languages, Formal Methods, Debugging, Simulation.

I. INTRODUCTION

Many tools have been proposed for the development of embedded software, in which the validation activity may represent more than 60% of the whole development effort. In this process, functional validation aims at checking that the system design is correct with respect to requirements, but few tools exist for the functional validation of system requirements themselves.

In this paper, we focus on real-time requirements, such as the following cruise control example:

"When active, the cruise control shall not permit actual and desired speeds to differ by more than 2 km/h during more than 3 seconds."

Such requirements usually describe a combination of logical and numerical properties of system signals over time. They stand in contrast to non-functional requirements, such as performance, usability, reliability, cost, etc. Being requirements, they express what a system should do or not do, but they do *not* describe how to achieve it: here for instance it could be done with a PID (proportional-integral-derivative) controller.

In practice, requirements are mostly written in natural language and are generally validated through manual reviews. As a consequence, many ambiguities and errors remain until validation testing. It is well-known that the later these errors are detected, the more expensive it is to fix the bug. The cost is even worse when third parties are involved as the extra process iterations involve specification and contractual changes.

Argosim STIMULUS addresses this issue of requirement early debugging and validation by providing two key features:

(I) Expresses real-time requirements and environment assumptions in a formal yet close to natural specification language;

(II) Generates and observes simulation results that satisfy requirements under environment assumptions.

The ability to formalize requirements in an easy-to-read language is a necessary condition for the approach to get acceptance by users, while the ability to simulate "what systems shall do" makes requirements validation possible while writing specifications, instead of delaying it to a later phase of the development process. This limits specification errors and ultimately reduces costs in the design phase.

In this paper we show how STIMULUS supports these claims by describing its technical foundations and by illustrating its use to formalize, debug and validate the requirements of a car automatic headlights controller which was provided to us by a Japanese software and tools vendor.

II. RELATED WORK

Several requirement engineering tools do exist, such as IBM DOORS. Compared to STIMULUS, they focus on requirement management and traceability, rather than validation.

Specification and simulation tools, like UML/SysML or Mathworks Simulink, aim at modeling and validating system design and architecture rather than high-level requirements. There are efficient at describing *how* a system should be implemented, but they lack the expressiveness to describe and to simulate *what* a system should do without describing the *how*. The case-study of this paper will clarify this point.

Formal methods tools, in particular model-checkers and proof systems, like the Rodin platform based on Event-B [1], provide expressive languages and exhaustive validation features. There are the tools to use to *prove* the consistency of a set of worked-out requirements, but less so to incrementally *debug* partial, possibly incorrect requirements and to *discover* missing requirements, for the two following reasons.

1. They can only detect formalized inconsistencies: they do not help the user to discover problems that he has not anticipated.

   Consider the cruise control example given in introduction. Before submitting this requirement to a model-checker or a proof assistant, the user has to define a relevant property to prove on it, which is not trivial. Moreover, there is no *a priori* guarantee that the chosen property can detect a mistake like omitting the absolute value operator when translating the condition "speeds should not differ by more than two km/h". In contrast, as we will see, the simulation feature of STIMULUS enables the user to discover unanticipated problems in requirements by observing simulation traces.

2. Model-checkers and proof assistants require fairly complete requirements to issue relevant results, that is, either a successfull proof of a non-trivial property on them, or a meaningful counter-example to it. This makes this approach hard to use in an incremental specification process where rough initial requirements are progressively refined.

The two approaches are actually complementary:

- STIMULUS enables to debug requirements and to validate them by simulation; its strength is its ability to quickly exhibit problems. This provides a level of confidence in the quality of requirements, which is arguably higher than the level provided by manual reviews, but do not deliver either a formal proof of consistency and correctness.
- Once good quality requirements are obtained, they can be submitted to model-checkers and/or proof systems to obtain formal proofs. Problems can still be discovered at this step, but much less frequently than if the previous step had not been performed.

This complementarity motivated a partnership with SafeRiver, a consulting company specialized in safety and cyber-security for software-based systems. SafeRiver makes intensive use of formal methods and tools, and is looking for solutions to speed up the correct formalization of safety requirements before performing exhaustive proofs on them.

III. STIMULUS TECHNICAL FOUNDATIONS

The scientific backgrounds of STIMULUS being detailed in [1], we give only an overview of it. The two key features of STIMULUS are an expressive formal specification language and a simulation engine based on a constraint solver.

A. A Constraint Real-Time Specification Language

Stimulus combines the concepts of the synchronous languages LucidSynchrone [5] and Lutin [6]:

- LucidSynchrone (like its industrial version SCADE) provides proven and mature concepts for modeling real-time systems, such as dataflow equations, hierarchical state machines, and synchronous parallel composition.
- Lutin provides the concepts needed for modeling real-time *non-deterministic* behaviours, namely dataflow constraints and non-deterministic control choices. It was designed for describing generic test scenarios for real-time systems.

We list below the main concepts of the resulting language.

a. Data: Synchronous Data-Flow Constraints

The behaviour of signals is specified with dataflow constraints like:

```
count = (0 -> last count) + (if evt then 1 else 0);
count <= 10;
```

in which the integer signal `count` counts the number of occurences (in time) of the Boolean signal `evt` and is constrained to be less than or equal to 10. `->` is the initialization operator: `e1 -> e2` evaluates to `e1` on reset, and to `e2` otherwise, and `last v` denotes the value of signal `v` in the previous step. In other words, this specifies than `evt` might be true at most ten times during an execution, but specifies nothing more about when `evt` can or cannot be true.

b. Control: Hierarchical State Machines.

They are typically used to model running modes, to define the temporal operators of the standard library, and also to model probabilistic choices in scenarios. States can contain constraints, as shown by the state machine of Fig 1. State machines in STIMULUS have a strong, preemptive transition semantics according to the terminology of [5]: transitions are fired and their condition evaluated in the same step as their destination state. In addition, they implement an original notion of termination which happens to be crucial to combine the temporal operators of the standard library.

c. Modularity: Systems and Macros.

As other synchronous languages or Mathworks Simulink, Stimulus encapsulates statements into *systems* that can then be

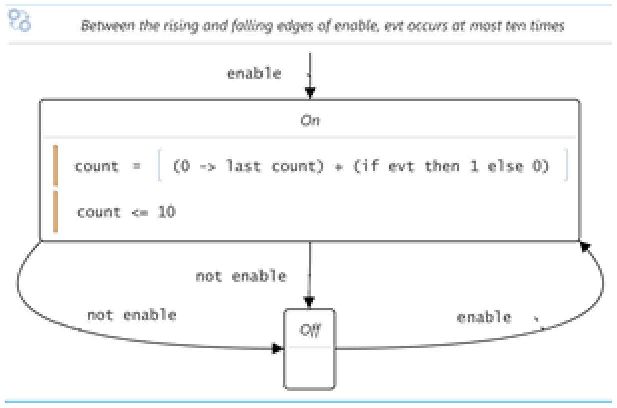

Between the rising and falling edges of enable, evt occurs at most ten times
enable
On
count = (0 -> last count) + (if evt then 1 else 0)
count <= 10
not enable
not enable
enable
Off

Fig. 1: requirement about the occurrences of evt

instantiated at several places. One can for instance define a system *Count* that counts the number of occurrences of an event (using an internal memory) and that can be reused to specify more complex constraints, like `Count(evt1) >= Count(evt2)`.

Stimulus also provides a robust system of *macros* with clear scoping and typing rules that accepts statements as parameters, in addition to signals having a value. They are used to define temporal operator, such as the

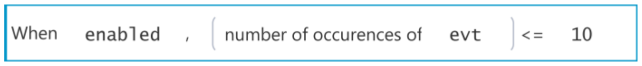
when <*condition*>, <*BODY*>

operator defined by the macro depicted on Fig. 2.

d. Readability: Sentence Templates

Formal requirements are easier to read when they look like textual requirements instead of programs. To achieve this goal, a system or macro can be associated with a user-defined *format* that specifies how instances should be edited and displayed in the editor.

For instance, if one associates to the macro *When* and the system *Count* the formats `when %condition%, %<BODY>%` and `number occurences of %event%`, the requirement of Fig. 1 rewritten using When and Count will appear as:



When enabled , number of occurences of evt <= 10

which is arguably more concise and more readable than the automaton of Fig. 1, while still enjoying the same unambiguous, formal definition.

e. Architecture: Block Diagrams.

Systems can be instantiated not only as a sentence, as described above, but also as blocks connected to other blocks in block diagrams, like in Mathworks Simulink, see Fig. 8. This enables developers to graphically describe the architecture of a system and to visualize the flow of information in it.

Other technical features of the STIMULUS language are a construct for controlling constraint propagation and orienting

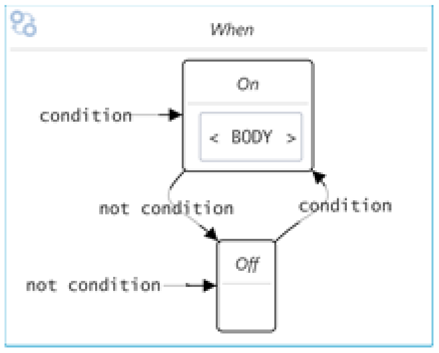


Fig.2 Macro defining the temporal operator *When*

constraints, in the spirit of [12], which provides scalability both for solving constraints and for model understanding by the user, type inference techniques of functional languages [13,14] to minimize the annotations required from users, and a physical dimension analysis [15,16] to statically detect this kind of inconsistencies.

Regarding the modeling of time, currently STIMULUS provides a simple periodic physical time model, by using a period to assign physical time values to logical time steps. In the long term we plan to provide a more flexible, aperiodic time model.

B. Compilation Process and Simulation Engine

SIMULUS compilation process follows the principles of [3,4,5,6] for (i) reducing parallel composition of hierarchical state machines to sets of statements guarded by choices on the *clocks* representing the active states of state machines, and (ii) ordering statements properly with respect to dependency constraints.

The simulation engine follows the principles of [7,8] and combines an exploration algorithm for resolving the non-deterministic choices induced by state machines and a constraint solver for resolving the non-determinism induced by constraints on variables. The solver handles logico-numerical constraints mixing logical operators on Boolean and enumerated variables, and linear constraints on numerical variables. The restriction to *linear* numerical expressions concerns only the unknown variables to be solved: the solver can deal with non-linear sub-expressions on variables that are known at solving time, like inputs or memories. For instance the constraint `x*(last x) >= y*(last y)*(last y)` is linear on the unknowns `x` and `y`.

IV. METHODLOGY FOR DEBUGGING REQUIREMENTS

In the previous section, we gave an overview of a new real-time constraint programming language built by combining the two research lines of work lead by M. Pouzet [3,4,5,6] and by the synchronous team of VERIMAG laboratory [7,8,9,10]. Now, how can it be used to debug requirements of real-time systems ?

With the current practice, requirements are mostly textual that are worked out and validated through manual reviews, and then given as inputs to

1. system designers and programmers on the one hand, who will implement the system;
2. test engineers on the other hand, who are in charge of writing functional test cases and test verdicts to confront the implementation w.r.t. the initial requirements.

Fig. 3 depicts the functional test bench architecture of a real-time system. I and O denote resp. the inputs and outputs of the System Under Test (SUT), which is considered as a black box. The box "Scenarios" feeds the SUT with inputs I, possibly taking into account outputs O to produce realistic inputs. The box "Requirements" reads both inputs I and outputs O of the IUT and emits a verdict.

As mentioned in the introduction, the experience shows that half of the bugs discovered by functional tests are requirements bugs, and not implementation bugs. The problem is that these bugs can be found only after the SUT becomes available.

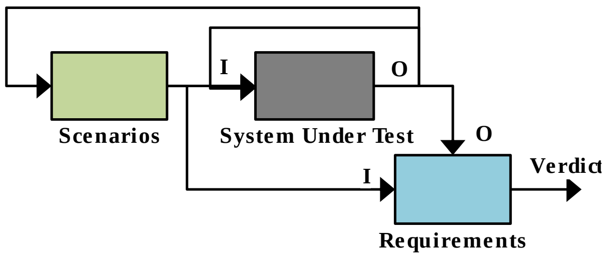


Fig. 3 : Classical test architecture of a real-time system

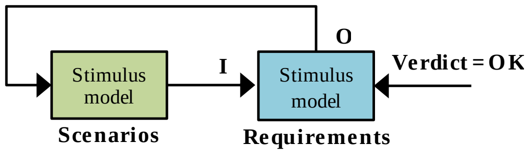


Fig. 4 : Debugging architecture for requirements and scenarios with STIMULUS

In contrast, the methodology made possible by the STIMULUS constrained-based language and its simulation engine is the following one:

- Requirements are seen as constraints between inputs I, outputs O, and the verdict (OK/NOK).
- The box "requirements" in Fig. 4 act as an outputs *generator*. At each simulation step:
    - the inputs I and the verdict OK are provided to the box "requirements";
    - the simulation engine solves the constraints in the remaining unknowns O and picks a random solution for them.
- Similarly, scenarios are also seen as constraints between I and O, which may describe general assumptions on inputs and/or more specific test cases.
- The box "scenarios" in Fig. 4 acts as an inputs generator: given outputs O provided to it by the feedback loop, the simulation engine solves the constraints in the remaining unknowns I and picks a random solution for them.

The fact that scenarios may be generic scenarios with a large variability on control and data allows the simulation engine to generate many different simulation traces, and ultimately makes more likely the discovering of an unexpected behaviour corresponding to a problem in the requirements. Actually scenarios are optional: one may simulate requirements alone. However, in practice, one often needs some general assumptions on the stability or the variation of signals to make simulation traces readable or realistic.

Another important observation is that once scenarios and requirements have been worked out using the architecture of Fig. 4, they can be reused directly in the test architecture of Fig. 3 with a black-box system under test: indeed, there are executable, and STIMULUS have a mechanism to turn a generator into an observer, which enables to turn the requirements box of Fig. 4 in the requirements box of Fig. 3.

We described in the two previous sections the scientific foundations of STIMULUS and the methodology it enables for debugging requirements together with their associated generic test scenarios. The sequel of the paper aims at answering to the following questions:

- How does it work in practice? Is it easy to formalize informal textual requirements with STIMULUS? How far is the formalized version to its informal counterpart in term of readability (traceability feature)?
- Are simulation traces effective at discovering problems? In other words, despite the non-exhaustiveness of our validation-by-simulation approach, is it "exhaustive enough" in practice?

To answer to these questions, we will illustrate the use of STIMULUS for formalizing and debugging requirements of an automatic light system coming from the automotive industry.

V. FORMALIZING THE REQUIREMENTS OF AUTOMATIC HEADLIGHTS

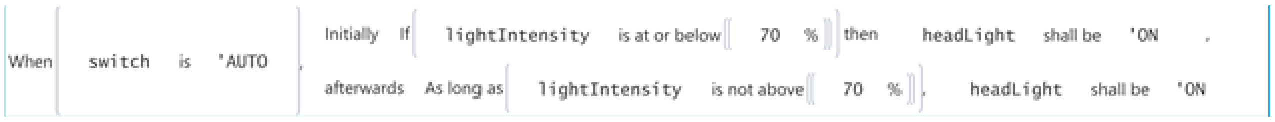
These requirements were provided to us by a Japanese software and tools distributor, as a typical example of the kind of requirements his customers have to deal with in the automotive industry. We insist on the fact that they have not been specifically invented for evaluating a tool like STIMULUS, neither by us nor by researchers or developers of alternative solutions.

The original, textual specification of the automatic headlights is depicted on Fig. 5. The head sentence is a sort of informal, high-level requirement that describe the general purpose of the four more precise, lower level requirements that follow. This purpose is to command the switching of the lights. In the sequel we will formalize the four requirements named **3Aa, 3Ab**, **3B** and **3C**.

Fig. 6 depicts the initial formalization of requirement **3Aa,** which will be debugged by simulation and upgraded in the

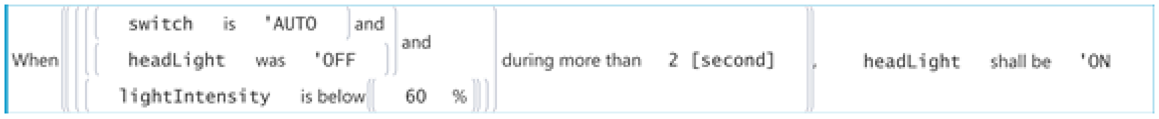
If the switch is AUTO then the headlights turn on or off, depending on the ambient light intensity - with a defined hysteresis to prevent blinking.
REQ_003Aa: if the switch is turned to AUTO, and the light intensity is at or below 70% then the headlights should stay or turn immediately ON.
Afterwards the headlights should continue to stay ON in AUTO as long as the light intensity is not above 70%.
REQ_003Ab: if the switch is turned to AUTO, and the light intensity is above 70% then the headlights should stay or turn immediately OFF.
Afterwards the headlights should continue to stay OFF in AUTO as long as the light intensity is not below 60%.
REQ_003B: if the switch is in position AUTO, the headlights are OFF, and the light intensity falls bellow 60%, then the lights should turn ON if this condition lasts for 2s.
REQ_003C: if the switch is in position AUTO, the headlights are ON, and the light intensity is above 70%, then the lights should turn OFF if this condition lasts for 3s.

Fig. 5: Original specification of the automatic headlights



When switch is 'AUTO , Initially If lightIntensity is at or below 70 % then headLight shall be 'ON .
afterwards As long as lightIntensity is not above 70 % , headLight shall be 'ON

Fig. 6 : Requirement **3Aa** in STIMULUS, initial version



When switch is 'AUTO and headLight was 'OFF and lightIntensity is below 60 % during more than 2 [second] , headLight shall be 'ON

Fig. 7 : Requirement **3B** in STIMULUS, initial version

next sections. One recognizes in Fig. 6 the pieces of sentence underlined on Fig. 5. This formalization is based on the use of STIMULUS standard library, which provides a number of sentence templates that are ubiquitous in real-time requirements, such as:

- "**as long as** <*expression*>**,** <*statement*>"
- "**initially** <*statement*>, **afterwards** <*statement*>"

As explained in Section III.A.d, such sentence templates are user-definable views for formal systems and macros, and their purpose is to make easy tracing the formalized requirement back to the textual, non-formalized requirement.

The other requirements **3Ab**, **3B** and **3C** are similarly formalized. **3Aa** and **3Ab** roughly specifies an hysteresis. **3B** and **3C** in addition require the light intensity to be low or high to hold for some time before switching respectively on or off the headlights. This is to prevent the headlights to blink too quickly, for instance when driving under a bridge.

VI. Simulating and Refining the Requirements

We just showed how the requirements can be formalized. However the major innovation of STIMULUS is the ability to simulate them.

A. *Simulation architecture*

Fig. 8 depicts the block diagram defining the simulation architecure considered in this paper. The block **Env** generates values for the signal **lightIntensity** that satisfy the assumptions depicted on Fig. 9. These assumptions combine general physical assumptions on the range of the light intensity (a percentage of a maximum intensity) and of its derivative, and a scenario making it alternatively increase or decrease. In this paper, we maintain the signal **switch** to **AUTO**, as we want to simulate the behaviour of the system under this mode.

The block **R003_v1** on Fig. 8 is defined by the requirements discussed in Section **Erreur ! Source du renvoi introuvable.**. This block will generate possible values for **headLight** that are compatible with these requirements and the values of the other signals.

B. *First simulation and detection of a problem*

Fig. 10 depicts a possible execution trace of the system defined by Fig. 8. One can observe the behaviour of **lightIntensity** and **headLight** signals. L60 and L70 are the two thresholds 60% and 70% that appear in the requirements.

What can be observed is that at start **headLight** has the expected behaviour: it is first OFF because the light intensity is above the 70% threshold, and when the light intensity falls below the 60% it becomes ON. However, afterwards the behaviour seems completely random and appears in contradiction with the intended behaviour.

Hence the simulation exhibits a problem either in the textual requirements, or in their formalization, or in both.

C. *Investigating and solving problem 1*

Consider the textual requirement **3Aa** on Fig. 6 and more precisely its second part:
"[…] Afterwards the headlights should continue to stay ON in AUTO **as long as** the light intensity is not above 70%".
The textual expression "as long as" is actually *ambiguous*: does it mean

1. **as long as** condition, something [**afterwards nothing**]" (sequential behaviour)

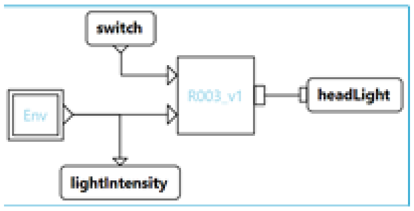
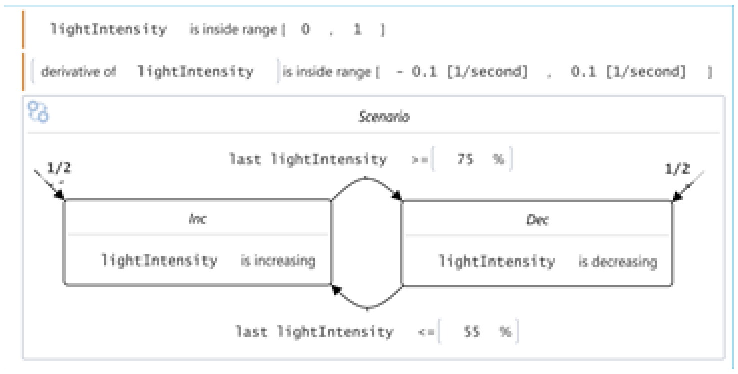


Fig. 8: simulation architecture



lightIntensity is inside range [0 , 1]

derivative of lightIntensity is inside range [- 0.1 [1/second] , 0.1 [1/second]]

Scenario

last lightIntensity >= 75 %

1/2 1/2

Inc: lightIntensity is increasing

Dec: lightIntensity is decreasing

last lightIntensity <= 55 %

Fig. 9: system *Env* describing the assumptions on the light intensity

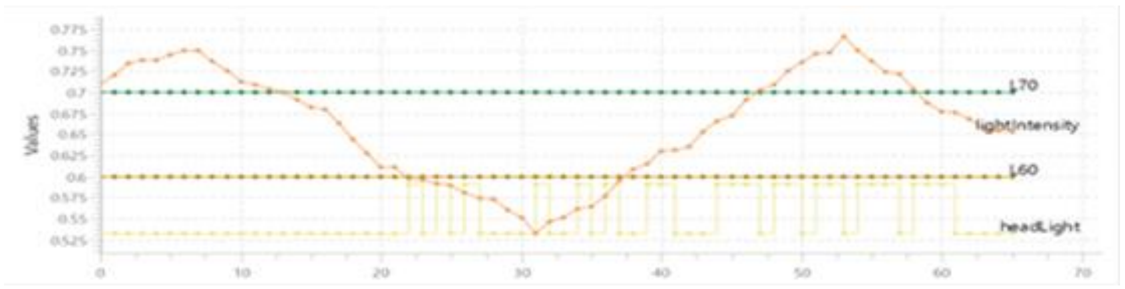
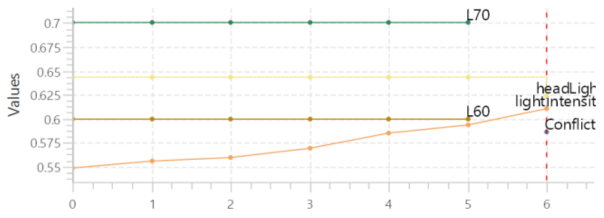


Fig. 10: Simulation of requirements, initial version

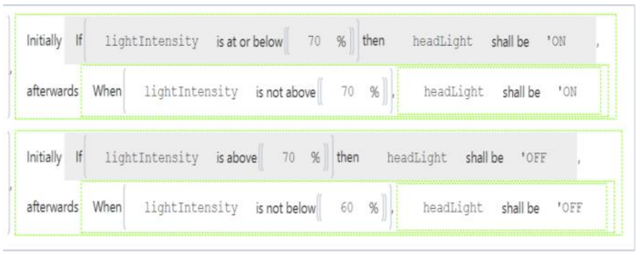
2. or “[**always**] **when** condition, something” (cyclic behaviour)?

This ambiguity is not really an artefact introduced by STIMULUS sentence templates: it is a real ambiguity which already exists in the textual version, between a sequential or a cyclic behaviour.

On Fig. 6 we opted inadvertently for the first interpretation, but clearly we expect a cylic behaviour here. Let us try the second one, which is available in the standard library. Requirement **3Ab** which follows the same pattern is similarly modified. Let us simulate this new version of requirements:



We obtain a *conflict* at simulation step 6: this means that some requirements are contradicting each other. Let us highlight the requirement **3Aa** and **3Ab** in the debugger at step 6 where the conflict occurs:



Initially If lightIntensity is at or below 70 % then headLight shall be 'ON ,
afterwards When lightIntensity is not above 70 % headLight shall be 'ON

Initially If lightIntensity is above 70 % then headLight shall be 'OFF ,
afterwards When lightIntensity is not below 60 % headLight shall be 'OFF

The debugger highlights the active parts of the requirements, and allows the user to discover that at this step, the requirements implie **headLight** to be ON and OFF at the same time!

D. *Investigating and solving problem 2*

Consider again the textual requirement **3Aa** on Fig. 1. We formalized the expression

“headLights should continue to stay ON”

underlined in Fig. 1 with the formalized sentence

“headLights shall be ON”.

Was that the right interpretation? The sentence could also mean:

“if it was ON, maintain it at ON, otherwise do nothing”.

To check this hypothesis, we defined a new, user-defined sentence template

“*<expression>* **should continue to stay** *<constant>*”

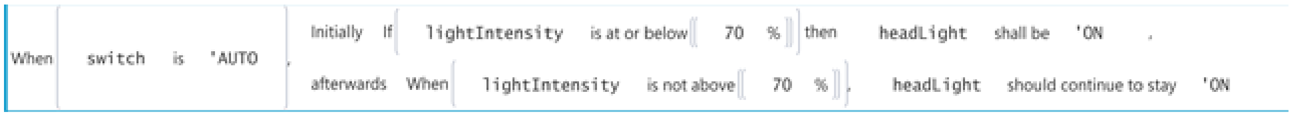

When switch is 'AUTO , Initially If lightIntensity is at or below 70 % then headLight shall be 'ON .
afterwards When lightIntensity is not above 70 % , headLight should continue to stay 'ON

Fig. 11 : Requirement **3Aa** in STIMULUS, second correction

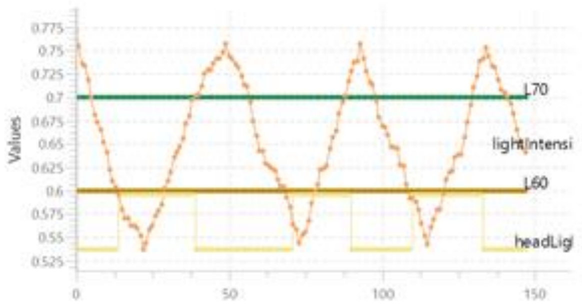


Fig. 12: Simulation of requirements after the second correction

with this semantics and used it to update requirement **3Aa,** resulting in the requirement of Fig. 11. Requirement **3Ab** was similarly modified.

Fig. 12 depicts a possible execution trace with this new version of requirements, which follows much better the expected behaviour of the head lights which should not blink and which should switch to ON or OFF according to two thresholds.

A last problem, that we will not detail as much, concerns the requirements **3B** and **3C**: they express that something should happen when a condition hold for some time duration, but say nothing about what should happen before. Hence unwanted, blinking head lights behaviour might occur with a more generic scenario in which the switch is not always equal to AUTO. This is a typical example of a missing requirement.

VII. Discussion

The four textual requirements of Fig. 5 look rather simple and reasonable. Yet they contain several ambiguities and omissions, some of them leading to unwanted behaviours, others to contradictions. Actually, although looking simple, they specify a complex behaviour that depends on the past history of signals, and on physical time. This complexity behind apparent simplicity is typical of real-time requirements and explains the strong need for their early validation, especially as they are often requirements of security-critical systems.

A. *Comparison of simulation approach to alternative requirements validation approaches*

It is difficult to discover by manual reviews the problems we discovered by simulation. This is why many of them are discovered later, either by developers or test engineers when they start exploiting the requirements given to them, or, worse, even later during the functional test phase.

We claim that model-checker or formal proof systems are not very suitable either for this *debugging and elicitation* task. For detecting and fixing the first problem found in Section VI.D, it is necessary

1. to explicitly formalize the higher-level property capturing the bad behaviour, which requires having already identified it as a potential error;
2. to check the validity of the requirements against this property – this will fail of course;
3. to analyse the cause of the failure and to try a new alternative.

The simulation approach made possible by STIMULUS

(i) completely removes the need for subtask 1; the user can still insert property observer to automatize the detection of an already identified potential problem, but this is optional;
this subtask is actually replaced by the observation of simulation traces enabling the *discovery* of unanticipated behaviours or getting an higher confidence in the relevance of the requirements

(ii) makes subtask 2 arguably easier: model-checkers may have an issue with too complex models, either in terms of size or expressiveness (non-linear computations for instance), and proof systems are not always fully automatic; in contrast simulation techniques are less computationally demanding and can handle complex models fully automatically[1].

Subtask 3 remains the same but in overall the removal or the simplification of the other subtasks make the trial-error cycle much quicker.

The use of model-checking and formal proof approaches is of course still meaningful to obtain an exhaustive confirmation of consistency, or to discover inconsistencies occurring in very uncommon cases that a simulation approach may miss.

B. *Evaluation w.r.t. announced criteria*

At the end of Section IV describing our methodology, we proposed two main evaluation criteria to our approach, namely:

1. Easiness of formalization, readability of formal models and traceability w.r.t. informal, textual requirement;
2. Effectiveness of the simulation engine in generating simulation traces exhibiting problems, that is, sufficient “practical exhaustiveness”.

[1] As mentioned in Section III.B, the constraints of STIMULUS models should be linear, but only on the unknown variables at solving time, which is not restrictive in practice.

We hoped to have successfully convinced the reader that the formalized requirements of Figs. 6 and 7 remain close to and as concise as their informal counterpart of Fig. 5. Although these formal requirements are not pure English, everybody can understand what they are talking about. In addition, they gain a perfectly formal semantics: the user can look at the formal definitions of the sentence templates to get a more in-depth understanding of a sentence, and simulate them to observe their dynamic behaviours.

Technically, this is a benefit of the combination of a powerful programming language (which is mostly hidden to the user), the use of sentence template as a view for programming constructs, and the design of well-thought standard library.

Regarding the second evaluation criteria, the two problems discovered in Section VI were both discovered with the very first simulation trace. This means that although the validation-by-simulation approach is not exhaustive, in practice the simulation engine is efficient at quickly exhibiting problems when they exist. In particular, the conflict discovered at the end of Section VI.C, which would have been easily found by a model-checker, was in practice as easily found by the simulation engine, using the very unspecific generic scenario of Fig. 9.

Of course, if the assumptions on the environment disallow some scenario, STIMULUS will not exhibit problems that are specific to them. But the same applies to model-checkers and formal proof systems: formal proofs are valid only w.r.t. the considered assumptions on the environment. STIMULUS actually supports the incremental process of starting debugging requirements with very simple scenarios, and later stimulating them with more complex or corner-case scenario. This process allows the user to progressively gain confidence in the quality of its requirements, instead of having to cope with too many unwanted behaviours at a time.

Technically, the effectiveness of the STIMULUS simulation engine for generating "really random" traces inside the set of possible traces satisfying the requirements comes from the use of a general constraint solver and of a fair algorithm to pick a random solution inside the solution space.

VIII. CONCLUSION

In this paper, we presented STIMULUS, a modeling and simulation tool for the early validation of functional real-time requirements, and we demonstrate its effectiveness for debugging a few requirements typical of the industrial practice, which contain subtle ambiguities and omissions.

STIMULUS is based on modern programming languages and simulation techniques, with many features dedicated to requirement readability. It exploits mature and well-proven research results to make things simple for the users.

It enables engineers to formalize requirements using predefined or user-defined sentence templates, to model environment assumptions and to observe execution traces that satisfy the requirements using the innovative simulation architecture of Fig. 4. It helps finding ambiguous, incorrect, incomplete, or conflicting requirements, as shown on the real-time requirements of an automatic head lights controller of a car provided to us by a software and tools vendor.

Relying on this case study, we discussed the additional benefits that the validation-by-simulation approach for requirements engineering proposed by STIMULUS can bring to the current validation-by-review and validation-by-proof approaches. One can actually observe than in the different domain of control system design, one of the most popular tool is TheMathworks Simulink, which also implements a validation-by-simulation approach.

We did not detail in this paper how to reuse STIMULUS scenario and requirements models for testing black-box real-time systems as depicted on Fig. 3, but this is a hot topic for our customers. Other topics are the automation of some common editing tasks, such as providing functional coverage criteria for requirements, and to automate debugging and testing tasks, such as guiding executions to favor the functional coverage of requirements according to these criteria.

REFERENCES

[1] Jean-Raymond Abrial: Modeling in Event-B - System and Software Engineering. Cambridge University Press 2010, ISBN 978-0-521-89556-9.

[2] Bertrand Jeannet, Fabien Gaucher: Debugging real-time systems requirements: simulate the "what" before the "how". EmbeddedWorld 2015.

[3] Grégoire Hamon, Marc Pouzet: Modular resetting of synchronous data-flow programs. Principles and Practice of Declarative Programming (PPDP'2000), ACM, 2000.

[4] Jean-Louis Colaço, Marc Pouzet: Type-based initialization analysis of a synchronous data-flow language. Journal on Software Tools for Technology Transfer (STTT), 6(3), 2004.

[5] Jean-Louis Colaço, Bruno Pagano, Marc Pouzet: A conservative extension of synchronous data-flow with state machines. Embedded Software (EMSOFT'2005), ACM, 2005.

[6] Dariusz Biernacki, Jean-Louis Colaço, Grégoire Hamon, Marc Pouzet. Clock-directed modular code generation for synchronous data-flow programs. Languages, Compilers , and Tools for Embedded Systems (LCTES'2008), ACM, 2008.

[7] Erwan Jahier, Pascal Raymond, Philippe Baufreton: Case studies with Lurette V2. STTT 8(6), 2006.

[8] Pascal Raymond, Yvan Roux, Erwan Jahier: Lutin: A Language for Specifying and Executing Reactive Scenarios. EURASIP J. of Embbeded System, 2008.

[9] Erwan Jahier, Nicolas Halbwachs, Pascal Raymond: Engineering functional requirements of reactive systems using synchronous languages, Symposium on Industrial Embedded Systems (SIES), IEEE, 2013.

[10] Erwan Jahier, Simplice Djoko Djoko, Chaouki Maiza, Eric Lafont : Environment model-based testing of control systems : cases studies. Tols and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 8413, Springer, 2014.

[11] The Mercury programming language. https://mercurylang.org/

[12] Zoltan Somogyi**:** A system of precise modes for logic programs. Int. Conf. on Logic Programming (ICLP'87), MIT press, 19878

[13] The OCaml programming language. caml.inria.fr

[14] The Haskell programming language. https://www.haskell.org

[15] Jean Goubault. Inférence d'unités physiques en ML. *Journées Francophones des Langages Applicatifs*, pages 3--20. INRIA, 1994.

[16] Andrew Kennedy: Dimension Types. European Symposium on Programming (ESOP'94), LNCS 788, Springer, 1994.

Incremental Life Cycle Assurance of Safety-Critical Systems

Julien Delange, Peter Feiler, Neil Ernst
Carnegie Mellon Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612, USA
{jdelange,phf,nernst}@sei.cmu.edu

***Abstract*—Finding problems and optimal designs in the requirements phase is more efficient than later phases. However, over-constraining the solution is also sub-optimal since not all information is necessarily available upfront. 'Build-then-test' approaches which insist on developing first requirements, then architecture, then implementation are not suitable for building systems that must be rapidly fielded and respond to ever-changing demands. Our approach, ALISA, is working on integrating four pillars for incrementally building systems which can be shown to satisfy the relevant requirements. Our four key pillars for assuring requirements satisfaction are requirements specifications, architecture models, verification techniques, and assurance case traceability between the first three. In this paper we introduce our approach, and highlight how we are integrating these pillars using an XText-driven DSL and tool meta-model leveraging existing tools and languages. Our current focus is on understanding exactly which requirements are responsible for the majority of design constraints. Identifying this subset promises to reduce architecture design space exploration and verification overhead, increasing delivery cadence.**

I. Introduction

Safety-critical systems function where an error can be mission- or life-threatening. They are carefully specified and designed according to a rigorous process, usually by different collaborating teams. Through the development process, system stakeholders define their goals, engineers define system requirements from them, and architects design the architecture, breaking the system into several layers and parts. Each part/layer is implemented by potentially different teams, tested separately and then integrated. One major issue of this actual process is the late discovery of errors: 80% of implementation errors are found at system integration but studies have shown that such issues (70% actually, shown in Fig. 1) are likely introduced earlier, when defining the system requirements. Thus, these errors could be discovered and fixed earlier, by improving requirements specification and design.

In this paper, we introduce our approach for improving requirements specification and design, called *Architecture-Led Incremental System Assurance*, ALISA. Our approach is traceable and testable from end to end, that is, from system requirements and stakeholders, down to software and verification outputs. This new method connects requirements to other system artifacts (specifications, models, code, etc.), enabling requirements traceability and validation along the development process. It provides assurance of requirements validation early in the development process, reducing certification cost of safety-critical systems through measurably better requirements and compositional verification evidence.

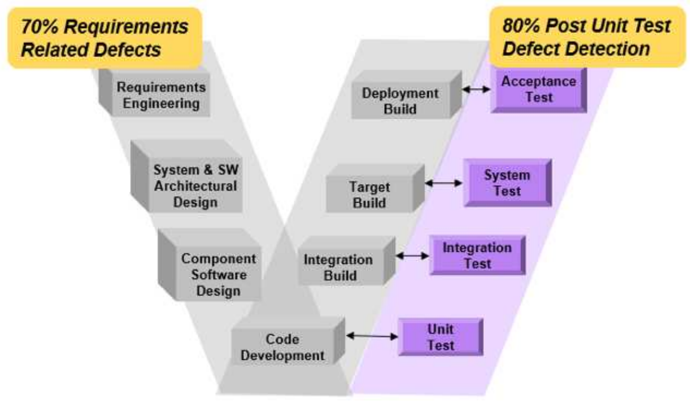


Fig. 1. The Double-V model, showing sources of errors

To improve the quality of requirements we focus on coverage of system specifications, quality attributes, and hazards, as well as management of uncertainty in the requirements. To improve the quality of evidence we use compositional verification, and multi-valued logic to automate the planning, execution of verification plans, and management, reporting of assurance evidence. In order to so we work with three different flavors of incrementality:

1) incrementality by refinement, working with one architecture layer or module at a time
2) incrementality by criticality, focusing on critical requirements/quality attributes first, and then the full set
3) incrementality by change impact, to manage the impact of changes on requirements, architecture design, and verification evidence.

II. Related Work

Moving from requirements to architecture is a key problem in software engineering. Current standards [1], [26] describe the life-cycle process to follow in order to develop software but consistency between development phases is often not synchronized, performed using a manual, labor-intensive process and consistency between each phase is not automated, and becomes out of date as products evolve.

One important issue has been to identify *architecturally-significant requirements* from a requirements specification; that is, those requirements which will have the most impact on how the system is implemented. For example, a well-developed methodology for this is the Quality Attribute Workshop [5] and design approaches such as the Architecture-Driven Design (ADD) approach. Seminal work includes moving from goals to agent-oriented software, with Tropos [14]; moving from KAOS specifications to software in [27]; and work by Dewayne Perry and his students [8]. One key advance is that there now exist many mature languages for both requirements (e.g., KAOS [9] and the URN standard [15]) and architectures (e.g., AADL [2] and SysML [22]). This allows us to leverage well-known formalisms for the translation. Furthermore, while there were hints in earlier work towards refinement, there was no explicit step for driving evidence-based changes in the requirements, as we propose in ALISA.

In the iterative context, requirements are ideally 'conversation starters' for design elaboration. For example, one takes the provided user story and queries the product owner about any uncertainties. And in an iterative context, particularly with iterations of 2-4 week duration, one can fairly easily refine these requirements. This approach does not work in all contexts, however, and may be guilty of finding local optima (e.g., under-designing), particularly in more complex systems [10].

The most similar approach to ALISA in integrating architecture and requirements is by AutoFocus 3 [4] and Whalen et al. [28]. Whalen et al. describe how SysML can be used with requirements models to accommodate the essential hierarchical nature of system engineering: a flow from more abstract (system requirements) to less (software requirements, then architecture models and code). It is key to recognize that there may be existing architectures and implementations that flow 'upward' to constrain requirements. This paper illustrated that it is often component interaction that provides most system failures, which can be traced to improper requirements decomposition and refinement. AutoFocus 3 [4], from the Fortiss Research Group, is a model-based engineering platform that, like ALISA, supports Eclipse-based end-to-end system development, starting with requirements and finishing with architecture. The two projects have a lot in common; key differences include the languages underpinning the architecture models (AADL in the case of ALISA, a custom language in for AutoFocus), and the verification mechanisms supported (nuSMV model checking in the case of AutoFocus; any AADL compatible verification approach in the ALISA case).

The concept of 'virtual integration' tries to leverage the promise of these model-centric approaches to reduce cost/cycle-time and risk (i.e., rework) by using early, and frequent, virtual integration, illustrated in the System Architecture Virtual Integration (SAVI) initiative ([23], [12]). For simplicity, a related paper [24] suggests doing requirements modeling directly in the architecture-modeling tool (in this case, Simulink). Although adoption and ease-of-use goals are important, our experience suggests this is not ideal for complex requirements models and higher layers of abstraction. For architecture-centric approaches the specific details of the 'architecture' is ambiguous. There are at least four types of architecture we have identified:

- *Functional architectures* capture functional requirements but with little or no information about how those functions will be encapsulated in components.
- *Conceptual architectures* specify how a system is decomposed into software and hardware components and the interfaces between them. Conceptual architectures are used during architecture trade studies and acquisition planning.
- *Design architectures* specify detailed performance characteristics of individual components, including internal design detail to the level required to support the analyses desired.
- *Implementation architectures* specify details needed to integrate and verify an overall system; for example, data that can be used to automatically generate configuration files or perform model-based testing.

This overlapping of abstractions makes it very difficult to properly separate solution context from problem context. Our intent is to provide separate languages (and vocabularies) for discussing these abstractions (for example, requirements specification tools for "functional architectures"), linked with shared identifiers.

In architecture-centric approaches, there is no explicit notation for capturing requirements (as part of a single process). For instance, the values for rate of change of speed thresholds are defined external to the modeling approach. This makes it unclear where and why these values are derived. For example, if the car we are building is a sports car, high acceleration may be desirable. If the car is a minivan (where small children may be more likely), high acceleration may be undesirable. To add traceability and rationale from requirements to architecture, we focus on linking stakeholder goals, system requirements, and an assurance case model for mapping verification strategies to goals. The compositional reasoning from [28] could be integrated as another technique for modeling architecture components and verification strategies.

III. Languages

Defining and verifying requirements rely on several concepts, that are addressed today by separate tools that are not integrated. Each tool covers one or several concepts but does not address the whole process from requirement definition to system validation. As shown in Table I, we distinguish the following concepts, which we call the four pillars of system integration [11]:

- **Requirements and Goal Definitions**: defines the stakeholders, system objectives and requirements. Requirements engineering frameworks offer a formal specification of system requirements, avoiding textual specification. There are several tools to capture and model requirements, such as KAOS [9] or RDAL [7].
- **Architecture Specification**: capture the system architecture structure. This is currently managed by languages such as SysML [22] or AADL [25].

- **Verification**: analyze system artifacts (i.e. model, code, specification), to check requirements. However, these activities are not directly related to the system requirements specifications. This is currently handled by model analysis tools, such as Resolute [13].
- **Claims, Arguments, Assurance**: show how the system enforce the requirements and provides confidence about system quality. This is currently handled by linking verification to requirements and architecture using Structured Assurance Cases (SACM) [21].

One (or several) pillars are supported by existing tools, but, as the sparseness of Table I shows, the artifacts are independent and loosely coupled, and thus, do not cover the entire development process, from requirements specification to system validation. In addition, some tools support the same pillars but with a different (and potentially inconsistent) approach.

For that reason, we propose to unify these concepts. Leveraging existing approaches, we address each pillar with a separate language, and connect them with tool support. Using such an approach, users can then specify their requirements, attach them to the architecture and ultimately, validate them, demonstrating system compliance with the requirements.

We defined the following languages:

- **ReqSpec**: stakeholder goals and system requirements. The language borrows concepts from RDAL [7] and KAOS [9].
- **Verify**: for verification activities, verification plans, methods (i.e. how to analyze and process system artifact to verify a property). This language is based on concepts from SVM [3], JUnit [17] and Resolute [13].
- **Alisa**: for defining assurance work areas, tasks This language borrows concepts from Mylyn [16].
- **Assure**: for assurance case instances. This language reuses concepts from JUnit [17], Resolute [13], SACM [21]

These languages have been implemented within Eclipse using the Xtext [6] framework. Our tool supports requirements/goals specification, validation methods and activities to check requirements enforcement in the architecture/implementation and automatic assurance case generation using the Goal Structuring Notation (GSN) [1] The tools check for requirements coverage (what architecture elements is missing a requirement, and vice versa), consistency (is there conflict between requirements) and auto-generate assurance cases using the GSN notation and tooling from [19] to show how requirements are validated and enforced within the architecture.

Note that we are not committed to a requirements $\rightarrow$ architecture mapping; indeed, in most cases we expect to have an existing architecture and requirements models, so the tracing can be either direction. We call this approach "architecture-led requirements specification" to capture the notion that the architectural model is the central hub of our model-driven engineering approach, with the spokes being the requirements, verification plans, and other artifacts. This does not demand that an architecture exist before requirements, but it does acknowledge that a strictly linear process is not realistic.

[1]http://www.goalstructuringnotation.info

IV. The Lightbulb Example

We show how to use ALISA languages and concepts on a simple system: a light-bulb being powered by a battery. The objective is that the battery has enough capacity to power the bulb.

The architecture of this system is shown in Figure 2 (the AADL textual representation is shown in Listing 1). It consists of two devices: one battery and one bulb. Both components are connected through a power socket. The goal of our example is to show that the battery has enough capacity to power the bulb.

While the graphical representation does not include AADL properties (that specify power capacity and budgets), the textual representation (listing 1) includes such information to capture the power capacity and budget.

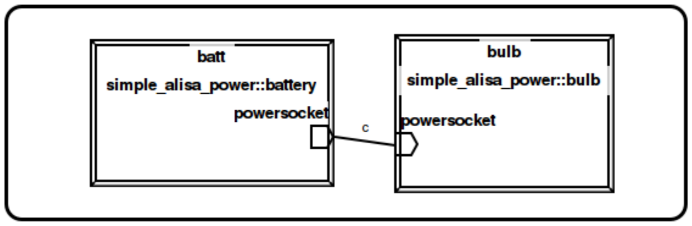


Fig. 2. AADL model of the bulb example

```
package simple_alisa_power

public

with SEI;

bus power
end power;

device bulb
features
   powersocket : requires bus access power;
properties
   SEI::PowerBudget => 60.0 W applies to powersocket;
end bulb;

device battery
features
   powersocket : provides bus access power;
properties
   SEI::PowerCapacity => 80.0 W;
end battery;


system integration
end integration;

system implementation integration.i
subcomponents
   bulb : device bulb;
   batt : device battery;
connections
   c    : bus access batt.powersocket -> bulb.powersocket;
end integration.i;
```

		KAOS	RDAL	Resolute	AGREE	SACM	AADL
Requirements	Stakeholder Goals	X					
	System Requirements	X	X				
Architecture	Specification	X			X		X
	Instance						X
Verification	Activities		X	X	X		
	Verification Methods		X	X	X		
Assurance	Claims and Arguments			X		X	
	Assurance Results			X		X	

TABLE I. TOOL COVERAGE AND GAPS IN SYSTEM PHASES

```
end simple_alisa_power;
```

Listing 1. Architecture of the lightbulb system

To specify the system requirements, the first step is to specify the stakeholders of the system, as shown in listing 2. For this example, we will keep the number of stakeholders to one, the system electrician.

```
organization mycompany
stakeholder electrician
 [ full name "John Doe" ]
```

Listing 2. Stakeholders definition

The next step consists in defining the stakeholders requirements, also known as goals (i.e. what the system is supposed to be and what constraints it is supposed to comply with). Goals definition are written using the **ReqSpec** language, as illustrated in listing 3. A goal is bound to a component (in the present example, the global system), a description and is associated with a stakeholder (the electrician defined before).

```
stakeholder goals mygoals for simple_alisa_power::integration
[ goal g1 : "Power OK" [
    description "We should be able to power the bulb"
    rationale "Without light, we cannot see"
    stakeholder mycompany.electrician
]]
```

Listing 3. Stakeholders Goals (REQSPEC lang.)

Once goals are defined, one should define the system requirements. The **ReqSpec** language is also used to define system requirements, as shown in listing 4. A system requirement is associated with a component, defines a description and is ultimately associated with a goal so that the tool is able to trace goals coverage (what goals are being linked to system requirements) but also architecture validation (what components have requirements).

```
requirement specification myrequirements
for simple_alisa_power::integration
[
   requirement enough_power : "The battery should have \
    enough power"[
       compute actualbudget
      description this "should have a battery with enough \
         power for the bulb"
      see goal mygoals.g1
   ]
]
```

Listing 4. System Requirements (ReqSpec language)

Then, verification plans are defined with the **Verify** language, that specifies how requirements are verified. The assurance plan is separated by claim being validated using analysis tools. Listing 5 shows that the system requirements previously defined are supported by a claim `c1` that checks that the system has enough power. This claim is verified by a verification activity (`mylibrary.electric_requirements`, defined in a general verification library) that will analyze the AADL model and ultimately, check the property values, making sure that the power provided by the battery is more than the power required by the bulb.

```
plan myplan for simple_alisa_power::integration.i [
  claim c1 for myrequirements.enough_power [
    assert all [  mylibrary.electric_requirements
   ] argument "The bulb has enough power"
  ]
]
```

Listing 5. Validation Plan (VERIFY lang.)

Ultimately, the overall assurance plan is defined using the **Alisa** language that defines verification plans are executed (ordering of tests). This is shown in Listing 6, which shows a definition of a basic assurance plan that executes the verification plan defined before (`myplan`).

```
alisa myplan
assurance plan power for simple_alisa_power::integration.i
[ assert myplan ]
```

Listing 6. Assurance Plan Workflow (ALISA lang.)

We have also added a function to automatically export the result of the verification process into an assurance case using the Goal Structuring Notation (GSN) [19]. The objective is to formalize the validation activities into a standardized notation. Our tool export these results into a GSN format that can be processed by the D-case assurance case tool [18].

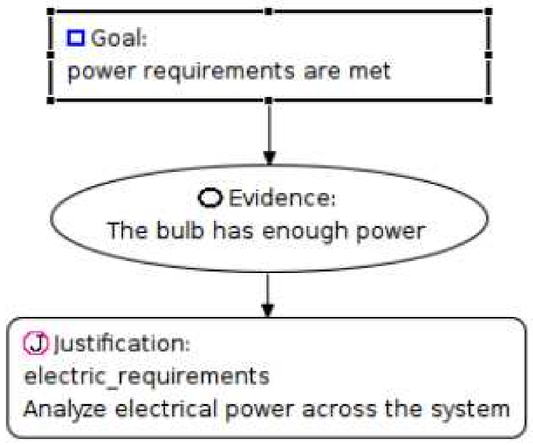

Goal:
power requirements are met
Evidence:
The bulb has enough power
Justification:
electric_requirements
Analyze electrical power across the system

Fig. 3. GSN export example

The GSN diagram of the bulb example is shown in Fig. 3. It details how the goal is decomposed and validated: the stakeholder goal (*Power requirements are met*) is decomposed into an evidence (*The bulb has enough power*) which are ultimately validated in the model (justification *Analyze Power Across The System*). Automating the production of the assurance case would avoid the labor costs associated with the production of such document, but also make them accurate with respect to the actual validation activities done on the model.

V. Larger-scale Example – The System Architecture Virtual Integration Case

In 2009 the SAVI initiative published a white-paper [12] describing a case study outlining how the notion of virtual integration – the use of an annotated architecture model as the single source for architecture analysis – can dramatically reduce rework and verification costs in safety-critical systems development. We have begun to implement that example in the ALISA tool-chain in order to demonstrate ALISA's suitability to realistic problems. In the 2009 report we modeled the sample problem – a Tier 1 airplane system - in AADL, one of the components of the ALISA tool-chain. We have completed this analysis using the ReqSpec tools, reverse engineering the requirements both from the existing architectural documents and pre-existing natural language requirements.

Figure 4 shows the SAVI Tier 1 model components, and figure 5 shows a portion of the Tier 2 model of the Integrated Modular Avionics subsystem. Note that AADL has a robust behavioral specification language, shown in Figure 6.

From these architectural models we focused on the flight guidance subsystem, creating appropriate requirements and verification specifications for these systems. The **Verify** language asserts that a given claim in the requirements (**ReqSpec**) is met. Listing 8 shows the verification plan for the requirements defined in listing 7. The Plugins.ResourceAnalysis verification activity (listing 8) calls a separate, standalone verification tool to check the associated requirement and ultimately returns one of {True, False, Unknown}. The automatic traceability support in our tool propagates the truth-value of each claim to mark requirements as satisfied (claims are True) or unsatisfied (claims are Unknown or False). The result will then be saved and used later to build the associated GSN (as the one shown in figure 3 for the lightbulb example).

We are currently researching better support for more complex propagation of verification outputs. For example, we may use semantics that provide for another verification activity if the first attempt is either Unknown (e.g. a model checker times out) or False (which we are calling fail-then semantics).

```
system requirements ADC_SW : "Requirements for the Software \
Subsystem of the ADC subsystem of the Flight Guidance System"

for Integrator::FGS::ADC::Spec::prAirDataFunction
[
   val UtilRatio = SystemConstants.UtilizationRatio

   val ADC_ProcessingBudget = SystemConstants.TBDm
   val ADC_RAMBudget = SystemConstants.TBDmb
   val ADC_ROMBudget = SystemConstants.TBDmb
   assert ADC_ProcessingBudget <=
     ADC_HW.ADC_ProcessingCapacity*Resource_Utilization_RoT
   assert ADC_RAMBudget <=
     ADC_HW.ADC_RAMCapacity*Resource_Utilization_RoT
   assert ADC_ROMBudget <=
     ADC_HW.ADC_ROMCapacity*Resource_Utilization_RoT

   requirement R1_1: "ADC Processing Budget" [
```

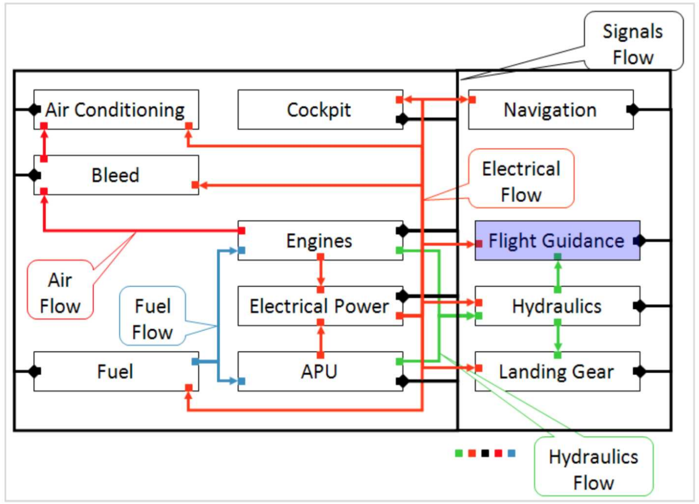


Fig. 4. Tier 1 (system view) of SAVI proof of concept

```
        description "The processing needs of the Software
           Subsystem of the ADC subsystem shall not exceed"
           UtilRatio "percent of"
           ADC_ProcessingBudget
    ]

    requirement R1_2: "ADC RAM Memory Budget" [
        description "The RAM memory needs of the Software
           Subsystem of the ADC subsystem shall not exceed"
           UtilRatio "percent of"
           ADC_RAMBudget
    ]

    requirement R1_3: "ADC ROM Memory Budget" [
        description "The ROM memory needs of the Software
           Subsystem of the ADC subsystem shall not exceed"
           UtilRatio "percent of"
           ADC_ROMBudget
    ]

]
```

Listing 7. FGS Requirements specified in ALISA's ReqSpec

```
verification plan ADC_SWPlan for ADC_SW
[

    claim ADC_SW.R1_1 [
        activities
           processingbudget: Plugins.ResourceAnalysis()
    ]

    claim ADC_SW.R1_2 [
        activities
           RAMbudget: Plugins.ResourceAnalysis()
    ]

    claim ADC_SW.R1_3 [
        activities
           ROMbudget: Plugins.ResourceAnalysis()
    ]

]
```

Listing 8. FGS Requirements specified in ALISA's Verify.

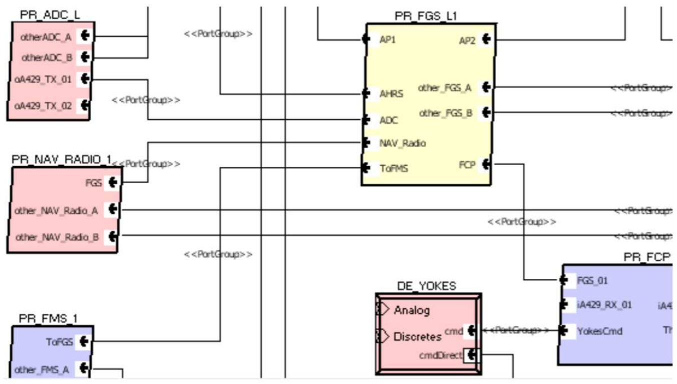


Fig. 5. Tier 2 AADL model of SAVI PoC

VI. Conclusions and Future Work

In this paper we presented ALISA, our vision for integrating four pillars for incrementally building systems: requirements specifications, architecture models, verification techniques, and assurance case traceability between the first three. We explained how we created DSL-based tooling to build these pillars, using a simple example. We then presented our example from an avionics domain to support our claim that the ALISA approach promises to reduce architecture design space exploration and verification overhead.

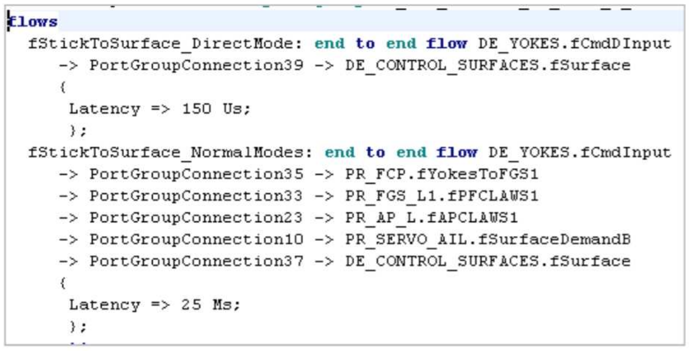
Our current and future work is to work with industry partners to add functional and safety requirements to an existing safety-critical system. We will apply the ALISA tools to perform compositional verification to provide assurance evidence. We are currently translating existing requirements from a DOORS and Excel environment to an ALISA-based requirements and safety hazards specification. We hope to demonstrate measurable improvement in requirements coverage and consistency. This will show the value of ALISA in early project phases. Our ultimate aim is to show measurable reduction in system rework costs by earlier defect detection. The certification process for safety-critical systems is one place where demonstrating compliance can produce large savings, so we are working with a collaborator to to produce additional evidence and certification artifacts to complement their testing evidence. Future work also includes integrating an assessment of requirements uncertainties (such as described by [20]), in order to further circumscribe the set of requirements that need to be checked. For example, in a car we may already be comfortable with our level of knowledge in the anti-lock braking subsystem, but less sure about the new fuel injector. We have also created import and export mechanisms with the OMG's Requirements Interchange Format (ReqIF)[2] in order to facilitate interchange.

Acknowledgments

Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of

[2]http://www.omg.org/spec/ReqIF/1.1/



```
flows
  fStickToSurface_DirectMode: end to end flow DE_YOKES.fCmdDInput
     -> PortGroupConnection39 -> DE_CONTROL_SURFACES.fSurface
     {
      Latency => 150 Us;
      };
  fStickToSurface_NormalModes: end to end flow DE_YOKES.fCmdInput
     -> PortGroupConnection35 -> PR_FCP.fYokesToFGS1
     -> PortGroupConnection33 -> PR_FGS_L1.fPFCLAWS1
     -> PortGroupConnection23 -> PR_AP_L.fAPCLAWS1
     -> PortGroupConnection10 -> PR_SERVO_AIL.fSurfaceDemandB
     -> PortGroupConnection37 -> DE_CONTROL_SURFACES.fSurface
     {
      Latency => 25 Ms;
      };
```

Fig. 6. Behavioral (flow) modeling in AADL

References

[1] Iso 15288 systems engineering—system life cycle processes. *International Standards Organisation*, 2002.

[2] S. Aerospace. *Architecture Analysis and Design Language (AADL)*, 2004.

[3] B. Aldrich, A. Fehnker, P. H. Feiler, Z. Han, B. H. Krogh, E. Lim, and S. Sivashankar. Managing verification activities using SVM. In *Formal Methods and Software Engineering*, pages 61–75. Springer, 2004.

[4] V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz. Autofocus 3: Tooling concepts for seamless, model-based development of embedded systems. In *Model Based Architecting and Construction of Embedded Systems (ACES-MB)*, 2015.

[5] M. R. Barbacci, R. J. Ellison, A. J. Lattanze, J. A. Stafford, C. B. Weinstock, and W. G. Wood. Quality attribute workshops (QAW). Technical Report CMU/SEI-2003-TR-016, Software Engineering Institute, 2003.

[6] L. Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.

[7] D. Blouin, E. Senn, and S. Turki. Defining an annex language to the architecture analysis and design language for requirements engineering activities support. In *Model-Driven Requirements Engineering Workshop*, pages 11–20. IEEE, 2011.

[8] M. Brandozzi and D. E. Perry. From goal-oriented requirements to architectural prescriptions: The Preskriptor process. In *International Software Requirements to Architectures Workshop (STRAW)*, Portland OR, May 2003.

[9] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):1–36, Nov. 2007.

[10] N. Ernst and G. C. Murphy. Case Studies in Just-In-Time Requirements Analysis. In *Empirical Requirements Engineering Workshop at RE*, pages 1–8, Chicago, Sept. 2012.

[11] P. Feiler, J. Goodenough, A. Gurfinkel, C. Weinstock, and L. Wrage. Four pillars for improving the quality of safety-critical software- reliant systems. Technical report, Software Engineering Institute - Carnegie Mellon University, 2013.

[12] P. H. Feiler, J. Hansson, D. de Niz, and L. Wrage. System architecture virtual integration: An industrial case study. Technical Report CMU/SEI-2009-TR-017, Software Engineering Institute - Carnegie Mellon University, November 2009.

[13] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen. Resolute: an assurance case language for architecture models. In *SIGADA Conference on High integrity language technology*, pages 19–28. ACM, 2014.

[14] P. Giorgini, P. Bresciani, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8:203–236, 2004.

[15] International Telecommunication Union. User requirements notation (urn). Recommendation Z.151 (10/12), ITU, 2012.

[16] M. Kersten and G. C. Murphy. Mylar: A degree-of-interest model for IDEs. In *International Conference on Aspect-oriented Software Development*, pages 159–168, Chicago, Illinois, 2005.

[17] V. Massol and T. Husted. *JUnit in action*. Manning, 2003.

[18] Y. Matsuno, H. Takamura, and Y. Ishikawa. A dependability case editor with pattern library. In *HASE*, pages 170–171, 2010.

[19] Y. Matsuno and S. Yamamoto. An implementation of GSN community standard. In *International Workshop on Assurance Cases for Software-Intensive Systems*, pages 24–28. IEEE Press, 2013.

[20] A. Nolan, S. Abrahao, P. Clements, and A. Pickard. Managing requirements uncertainty in engine control systems development. In *International Requirements Engineering Conference*, pages 259–264, Trento, August 2011.

[21] Object Management Group. Structured assurance case metamodel (SACM). Specification formal/2013-02-01, Object Management Group, 2013.

[22] OMG. Systems Modeling Language, 2006.

[23] D. Redman, D. Ward, J. Chilenski, and G. Pollari. Virtual integration for improved system design: The AVSI system architecture virtual integration (SAVI) program. In *Analytic Virtual Integration of Cyber-Physical Systems Workshop, 31st IEEE Real-Time Systems Symposium (RTSS 2010)*, 2010.

[24] N. Rungta, O. Tkachuk, S. Person, J. Biatek, M. W. Whalen, J. Castle, and K. Gundy-Burlet. Helping system engineers bridge the peaks. In *TwinPeaks Workshop at ICSE*, Hyderabad, India, 2014.

[25] SAE International. *AS5506 - Architecture Analysis and Design Language (AADL)*, 2012.

[26] R. Singh. International standard iso/iec 12207 software life cycle processes. *Software Process Improvement and Practice*, 2(1):35–50, 1996.

[27] A. van Lamsweerde. From system goals to software architecture. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures*, pages 25–43, Bertinoro, Italy, September 2003.

[28] M. W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. P. Heimdahl, and S. Rayadurgam. Your "what" is my "how": Iteration and hierarchy in system design. *IEEE Software*, Mar/Apr:54–60, 2013.

Session 20

Modeling for safety

Thursday 28th, 15:00 – Ariane 2

Aspect-oriented Data and Safety Modeling for Cyber-Physical Systems in Process Automation

Dirk Kuschnerus
Institute of Electronic Circuits
Ruhr-Universität Bochum
Bochum, Germany
Email: dirk.kuschnerus@rub.de

Attila Bilgic
KROHNE Messtechnik GmbH
Duisburg, Germany
Email: a.bilgic@krohne.com

Thomas Musch
Institute of Electronic Circuits
Ruhr-Universität Bochum
Bochum, Germany
Email: thomas.musch@est.rub.de

***Abstract*—Cyber-physical systems (CPS) integrate computation with physical processes, enabling the dynamic adaption of systems based on economic and environmental conditions. The adoption of CPS in industrial process automation is impeded by legacy systems with severe functional safety constraints and the need for highly configurable devices. To transfer the benefits of CPS to process automation, the inherent conflict between CPS safety and configurability must be explicitly considered during system design and operation. This paper proposes aspect-oriented modeling of safety and data for CPS in process automation as a baseline for formal consistency monitoring.**

I. Introduction

Cyber-physical Systems (CPS) integrate computation and physical processes [1]. A remaining key research challenge for CPS is the definition of suitable modeling approaches covering abstraction and functional safety [2], [3]. Devices used in process automation CPS are usually mass market products which must be adapted to the specific process environment of the customer. Suppliers implement this requirement by adding configurability to their products, tailoring them to the customer's requirements during production and on-site. This drastically increases device complexity, an effect which is amplified by the emergent requirements for highly dynamic systems arising from initiatives like the German "Industry 4.0", where factory automation shall enable the dynamic adaption of the production to customer's desires or real-time demand and economical factors along the life cycle of a product [4].
Unlike in factory automation, systems in process automation are typically large scale plants with complex static control loops, where despite its benefits CPS adoption is limited by functional safety restrictions imposed by standards such as IEC 61508 [5] due to risks for humans and environment. Safety-critical applications require extended verification of the plant's safety functions, which is impeded when the implementing CPS components are configurable, creating a dependence between safety-critical behavior and configuration data. This conflict raises the need for formal specification and verification techniques to enable safety verification of devices considering specific configurations. In [6], we introduced a domain model for hierarchical modeling of CPS in process automation. In this paper, we propose a modeling approach advancing our domain model, using the aspect-oriented paradigm to explicitly model the safety and data concerns separated from the CPS domain model. Our goal is to enable the reasoning about safety in presence of complex configurability, setting a baseline for the application of dynamic CPS in process automation while preserving the required functional safety of the legacy installations.
The remainder of this paper is organized as follows: Section II examines related work on CPS modeling, Sections III and IV describe the interconnected modeling of the safety and data aspects. Both aspects are jointly examined in a case study in Section V, which also covers inconsistencies between aspect and domain models. Section VI gives a conclusion and outlines further research directions.

II. Related Work

To obtain an overview of methods for modeling CPS in process automation, we conducted a systematic mapping following the method introduced in [7]. The mapping identified and categorized a total of 448 relevant publications, 15 of which target the domain of process automation. This low number is also confirmed by [8], where only one publication from the domain of industrial automation is listed.
Following our mapping, we further examined the relevant papers from all domains in a systematic literature review as suggested in [9], focusing on the aspects of data and functional safety. From the 15 process automation papers only [10], [11] and [12] address functional safety. A combination of safety and data modeling was only found in [12], where the reachability of unsafe hybrid parametrized automata states is determined but further safety and data concepts such as hierarchical modeling and data dependencies are not covered.
To additionally cover cross-domain and generic approaches, we broadened our investigation to relevant approaches from all domains and identified 5 further papers covering data and safety aspects in CPS ([13], [14], [15], [16], [17]). [14] describes architectural views for heterogeneous CPS models and consistency considerations between these views as well as behavioral semantics for system verification. The approach uses automata states to model and verify the safety of the CPS. [16] develops a formal framework and graphical notation for the development of hybrid systems using graphs and hybrid automata. Neither [13], [14], [15] nor [16] include the modeling of safety concepts that are incorporated in the design of the whole CPS on various abstraction levels or the in-depth coverage of configuration and the effects of configuration data on system behavior and safety. [17] does not cover physical and deployment views of CPS which we consider important for safety and data consistency. In addition, none of the publications mentioned before use aspect-oriented modeling

for the cross-cutting concerns of safety and data.
Concluding our literature review, to the best of our knowledge our approach differentiates from the state of the art by covering the explicit aspect-oriented modeling of functional safety and data for CPS in process automation. We are further not aware of generic or cross-domain approaches providing a comprehensive modeling coverage of safety and data concepts which we consider crucial for CPS in process automation.

III. Aspect-oriented Safety Modeling

In aspect-oriented development, cross-cutting concerns affecting major parts of the development artifacts are represented as detached aspects to achieve separation between functionality and cross-cutting concerns [18]. During the definition of our domain model, we identified the influence of functional safety on its *structural*, *behavioral*, *physical*, *deployment* and *communication* viewpoints on the *plant*, *CPS*, *subsystem* and *component* layers of abstraction. In addition to their distributed influence, safety concepts typically are major development artifacts which regarding to their specification and certification efforts are desired to be reused in multiple projects. We therefore follow the aspect-oriented paradigm by integrating these concerns into safety aspect models abstracted from a specific device development and connecting them to our domain model using formal weaving functions.
Fig. 1 shows the scope of our safety aspect. Aspect models

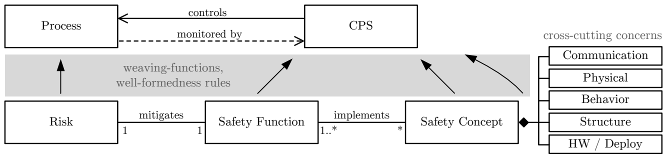


Fig. 1. Scope of the safety aspect

define cross-cutting safety concerns from all viewpoints and abstraction layers. Multiple concerns are integrated to safety concepts implementing safety functions, which are defined in IEC 61508 [5] as functions controlling a process or system to mitigate the risk of dangerous failure causing harm to humans and environment. The connection points between aspect models and the domain model are formally defined by weaving functions and structurally constrained by well-formedness rules (*wfrs*). The following paragraphs give an overview of the aspect beginning with the cross-cutting concerns.
The influence of functional safety on the logical CPS structure is primarily specified by the criticality and safety integrity level (*SIL*) of structural model elements, needed as baseline for further approaches such as partitioning and deployment. The function *crit* defines whether an element is safety-related (*sr*), non-safety related (*nsr*) or has a mixed criticality:

$$\begin{aligned}\boldsymbol{crit}\ :& Entity \cup DeployContext \cup Channel \cup Protocol \\ & \rightarrow \{sr, nsr, mixed\}\end{aligned} \tag{1}$$

The function *sil* defines a SIL for each element used to determine its valid dependencies regarding safety. To connect this safety information to our domain model, we define the weaving-function *addAttribute*:

$$\begin{aligned}\boldsymbol{addAttribute} =& \{f_{aA_1}, ..., f_{aA_n}\} \\ f_{aA_i} =& E' \subseteq Entities \rightarrow type(att_i \in Attributes)\end{aligned} \tag{2}$$

specifying a function f_{aA_i} for each attribute A_i that shall be added to the domain model. f_{aA_i} assigns the type of A_i to each of the targeted Entities E'. f_{aA_i} is by convention named after the added attribute A_i. The application of every weaving function is constrained by associated structural wfrs.
IEC 61508 differentiates between systematic failures and random hardware failures e.g. due to hardware aging. The latter are included in safety analysis such as Failure Modes Effects and Diagnostic Analysis (FMEDA) to determine the risk of dangerous undetected failures of the safety function. We add hardware failure information to the hardware elements in the domain model using the weaving function $addAttribute$, covering the probability of a hardware error, the fractions of safe, unsafe, detected and undetected errors and aggregated values for system integrators.
The dependable transfer of safety-related data must be ensured by safe communication channels in safety-critical CPS. The applicability of a channel for transporting data of a specific SIL is influenced by the bit error probability of the underlying hardware link as well as qualitative and quantitative communication measures modeled by the safety aspect both in the *deployment* and *communication* viewpoints. Safety-critical CPS typically adapt their behavior according

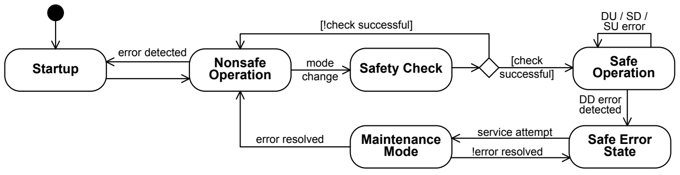


Fig. 2. Safety modes introduced by the safety aspect

to their internal safety mode. To model the overlaying safety modes of the CPS under development and the modification of the domain model behavior during the modes, our aspect uses a safety automaton connected to the behavioral semantics of the domain model via weaving functions. Behavioral changes triggered by the safety mode are the augmentation of the behavioral semantics, e.g. the addition of an additional safety check, and their restriction, e.g. due to a simulation function which may only be started in nonsafe operation. Both augmentation and restriction can be applied to all behavioral models used in [6]. Fig. 2 shows the generic safety automaton, describing the internal safety mode of the CPS from startup to nonsafe and safe operation. The CPS stays in safe operation in case of dangerous undetected and safe errors, switching to a safe state on detection of critical errors. The augmentation of behavioral semantics is used in the motivated verification of process safety functions considering their configurable realization. On the subsystem layer, safety functions are modeled by adding control modes to the hybrid automaton of the process step. The process safety time, i.e. the time between the occurrence of a failure and a resulting hazardous event defines the detection and reaction time available to the safety functions for mitigating the risk of the event and is modeled inside the safety automaton of the specific CPS. The realization of safety functions is modeled on the component layer by modifying the system's activity diagrams. Both concepts are detailed in section V-D.
As shown in Fig. 1, our safety aspect additionally defines reusable safety concepts from sets of cross-cutting concerns. These templates can e.g. define diagnostic functions to detect the hardware failures introduced above, facilitating the tracing

between failure analysis and mitigation as shown in the 2-channel safety concept in Fig. 11.

IV. Aspect-oriented Data Modeling

The device development for industrial CPS targets mass markets where it is unfeasible to develop individual products for every customer. As a result, complementing the measurement data the CPS gathers and processes, industrial CPS contain a large number of configuration data used for the customization of base device variants to the customer's application. Both data categories influence all viewpoints over the entire system hierarchy. A configuration data model is often used detached from a specific device, e.g. during manufacturing and order processes or customer service and should be separated from other development artifacts. We therefore model CPS data as aspect to enable data development, deployment and analysis to be conducted independently from the domain model.

Fig. 3 shows the fundamental data aspect model for CPS in process automation, categorizing data items into device variables and static data. Device variables represent values connected to the physical process which are periodically refreshed and distributed throughout the CPS, whereas static data items describe typically persistent attributes of the CPS itself. Every data item is uniquely identified by its *localId* and scope, which defines the area where device variables are distributed and static data is synchronized and available to all modules. This can be e.g. a partition or a node. Device variables are produced by one distinct software module (*producer*), can be set to a synthetic test value (*synthetic*) and connected to an external CPS interface (*outputChannel*). Each device variable has an update interval and can be kept local by stopping the bus distribution. The data aspect defines two types of device variables: a float value with timestamp and a double value. Static data is protected by an access level specifying valid editors as well as the item's settings for persistent storage and replication between data stores. Static data is defined for several data types and contains actual, default and SIL values and associated value ranges.

The data aspect also defines dependencies between data items. Configuration details of device variables such as limits and default values are given by a referenced static data item (*configuredBy)*. Both device variables and static data can be derived from other data items. Derived data items are locally calculated from primary data items and are not persistently stored. In addition, static data items can interfere with each other during the update of their values. An *updateDependency* between multiple data items denotes that the CPS can only accept an updated value of a data item if the dependent items are also updated. The *checkDependency* is an internal dependency that triggers a validation of connected data items if the value of one data item is altered.

While the safety and data aspects are specified concurrently to the domain model, there is interdependence between the aspect models. Device variables and static data both have a criticality and SIL defined via the weaving function *addAttribute* from the safety aspect. In addition, the SIL value in static data items defines a value that is set when the safe operation mode specified by the safety aspect is entered.

The aspect model is connected to the domain model using

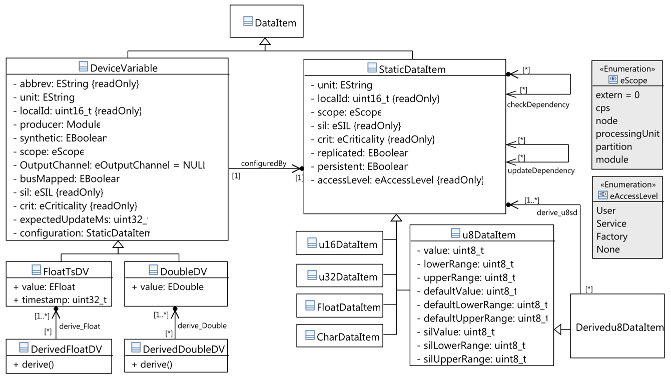


Fig. 3. Basic data model defined by the data aspect

the concepts of weaving functions and restrictions specified by wfrs as described in Section III. In the structural, communication and deployment views, attributes can be altered and entities such as software modules can be deactivated by the data aspect. As the data aspect models the CPS internal representation of process values, it adds a connection between process values and device variables thus connecting physical and computational parts of the domain model. In the deployment and communication views, the storage and distribution data flow of device variables and static data is modeled using the weaving function *addData*. This function also specifies hardware configuration by connecting static data items to hardware nodes. As described in Section III, the aspect models influence the behavior of the CPS. The influence of configuration data to the behavioral semantics is modeled using the weaving functions *addBehavior* and *removeBehavior* as shown in Fig. 9.

V. Case Study

A. Example Process - Distillation in MDI Production

We use the simplified chemical process of methylene diphenyl diisocyanate (MDI) production [19] as case study for exemplary application and evaluation of our modeling approach. MDI as a base product of polyurethanes is one of the most produced isocyanates. It can be generated by condensation of aniline and formaldehyde to methylenedianiline (MDA) using hydrochloric acid (HCl) as catalyst followed by phosgenation of the MDA. Using the highly dangerous phosgene, the process is well-suited to study our domain model in a safety-critical environment. As shown in Fig. 4, phosgene and HCl must be separated from the crude MDI after the phosgenation in a distinct subprocess "phosgene separation", on which we focus in our case study.

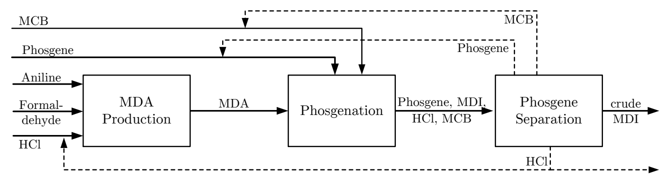


Fig. 4. Schematic Overview of MDI Production

The separation of phosgene, HCl, MDI and the solvent chlorobenzene (MCB) can be realized by distillation, as schematically shown in Fig. 5. The feed mixture F enters the column in liquid state and flows towards the base of the column which is continuously heated by circulation of the base product (MCB and MDI) through a steam reboiler. This leads to vaporization of phosgene, HCl and parts of the MCB, forming a vapor flow V towards the head of the column. This head product exits the column and enters a condenser, which cools down the vapor to liquefy MCB and phosgene while extracting the gaseous HCl. MCB and phosgene are then stored in an output tank and directed towards other subprocesses. In a non-ideal distillation column, the vapor emerging at the column bottom contains both the lighter and heavier compound. To reduce this mixing and to obtain higher purity of base and head product, column floors and a partial reflux L from the output tank to the column are installed. When the steam meets with the reflux at a certain floor, it partly condenses whereat mostly the heavier compound is liquefied. The condensing energy contrarily causes the vaporization of the lighter compound in the reflux. Moving towards the column head, this effect is multiplied by introducing additional floors to reach a desired level of purity. Typical distillation columns are controlled using five control

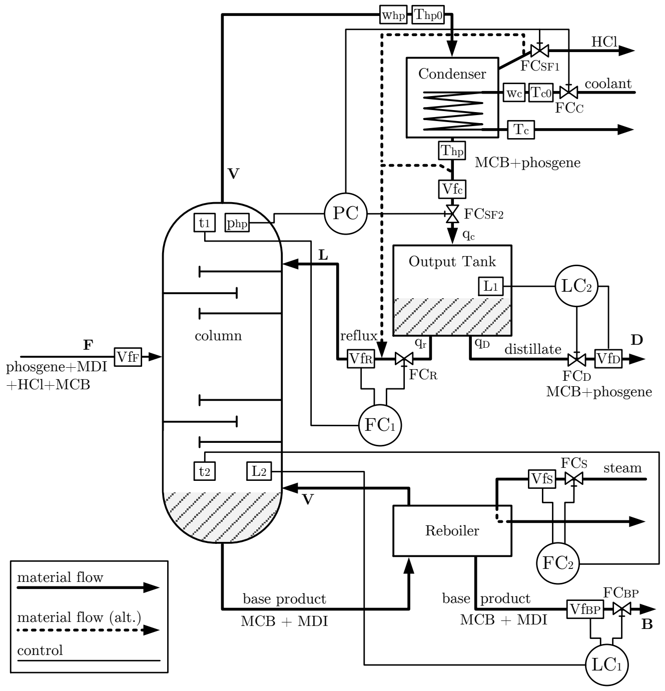


Fig. 5. Schematic Overview of Phosgene Separation by Distillation

loops. To obtain stable column operation, the levels at the column base (LC_1) and the output tank (LC_2) as well as the column pressure (PC) are controlled. The product quality is defined by the desired separation between the compounds in both base and head product (distillate). Control schemes for distillation columns are commonly named after the variables used to control the product quality. In our case study, we apply the LV control scheme, which controls the head product composition by the reflux (FC_1 controlling L) and the base product composition by the flow of steam to the reboiler (FC_2 controlling V). To reduce the size of our case study, we evaluate our aspect-oriented modeling approach focusing on condenser and output tank of the distillation. In the following section, we introduce both process steps and derive a state space representation to which we apply our approach.

B. State Space Representation

Control algorithms in process automation are designed based on system models either deduced from physical characteristics of the controlled process or aggregated from measurements. The variables and detail of the model must be adapted to the actual control task. In our case study, the safety-relevant criterion of the condenser is the complete liquefaction of phosgene, warranted if the condenser cools the head product lower than the boiling point of phosgene at 7.44 degrees Celsius. The controlled variable in the condenser is the mass flow w_c of the coolant. We abstract the behavior of the condenser with that of a heat exchanger:

$$\begin{aligned}\dot{T_{hp}} &= \frac{w_{hp}}{m_{hp}}T_{hp0} - \frac{w_{hp}}{m_{hp}}T_{hp}(t) - \frac{kAT_{hp}(t)}{m_{hp}c_{hp}} + \frac{kAT_c(t)}{m_{hp}c_{hp}} \\ \dot{T_c} &= \frac{w_c}{m_c}T_{c0} - \frac{w_c}{m_c}T_c(t) + \frac{kAT_{hp}(t)}{m_c c_c} - \frac{kAT_c(t)}{m_c c_c}\end{aligned} \tag{3}$$

where $\dot{T_{hp}}$ and $\dot{T_c}$ are the time-derivatives of the head product and coolant temperatures after condensing, k is a constant of the heat exchanger describing its ability to transfer energy, A is the area of the heat exchanger, m_{hp} and m_c are the masses, c_{hp} and c_c the heat capacities, w_{hp} and w_c the mass flows and T_{hp0} and T_{c0} the temperatures at the entry point of the condenser of the head product and coolant, respectively.
The change in head product temperature depends on the heat transfer between product and coolant as well as on their initial temperatures. For controlling the head product temperature by manipulating the mass flow of the coolant, we rewrite the second equation for $T_c(t)$ and insert it into the first to obtain a non-linear equation for the dependency of $\dot{T_{hp}}$ and w_c. To facilitate control and verification activities we generate a linear state space representation of the form

$$\begin{aligned}\Delta\dot{x_t} &= A\Delta x(t) + B\Delta u(t) + E\Delta d(t) \\ \Delta y(t) &= C\Delta x(t) + D\Delta u(t)\end{aligned} \tag{4}$$

where $\Delta x(t) = \begin{pmatrix}\Delta T_{hp}(t) \\ \Delta T_c(t)\end{pmatrix}$ is the state, $\Delta u(t) = \Delta w_c(t)$ the input, $\Delta d(t) = \begin{pmatrix}\Delta w_{hp}(t) \\ \Delta T_{hp0}(t)\end{pmatrix}$ are the disturbance variables and $\Delta y(t)$ is the output of the condenser. Δ symbolizes small deviations from the original variables in the cause of linearization. The matrices A, B, C, D and E are obtained by Taylor linearization of the equations in (3) at a specific operation point in which T_{hp} and T_c are stable, i.e. $\dot{T_{hp}} = 0$ and $\dot{T_c} = 0$:

$$\begin{aligned}A &= \begin{pmatrix}0 & -\frac{w_{hpR}c_c w_{cR}}{kA} - w_{hpR} - \frac{c_c w_{cR}}{c_{hp}} - \frac{kA}{c_{hp}} \\ \frac{kA}{m_c c_c} & -\frac{w_{cR}}{m_c} - \frac{kA}{m_{hp}chp}\end{pmatrix} \\ B &= \begin{pmatrix}-\frac{w_{hpR}c_c T_{cR}}{kA} + \frac{w_{hpR}c_c T_{c0}}{kA} - \frac{c_c TcR}{c_{hp}} + \frac{c_c T_{c0}}{c_{hp}} \\ T_{c0} - T_{cR}\end{pmatrix} \\ C &= \begin{pmatrix}1 & 0\end{pmatrix} \\ D &= 0 \\ E &= \begin{pmatrix}T_{hp0R} - \frac{c_c w_{cR}T_{cR}}{kA} - T_{cR} + \frac{c_c w_{cR}T_{c0}}{kA} & \frac{w_{hpR}}{m_{hp}} \\ 0 & 0\end{pmatrix}\end{aligned} \tag{5}$$

Variables with the index *R* are fixed in the operation point. While more complex models for condensing processes exist (see e.g. [20]), we use this abstracted form to keep the size of the verification problem manageable for our case study.
The output tank is the second process step covered by our case study. The critical process variable concerning functional safety is the level of the distillate $h(t)$ inside the tank which is controlled by LC_2 via the valve FC_D. The level is influenced by the input q_c from the condenser, the reflux q_r to the column and the output flow q_D:

$$\begin{aligned} q_c &= Vf_c * d_c(t) \\ q_r &= k_r * d_r(t)\sqrt{2gh(t)} \\ q_D &= k_D * u(t)\sqrt{2gh(t)} \end{aligned} \tag{6}$$

The flows are defined by the opening of the corresponding valve ($d_c(t), d_r(t), u(t) \in [0..1]$) and the constants k_r, Vf_c, k_D defining the maximum flow at the entry and exit points of the flow. For the level control, $d_c(t)$ and $d_r(t)$ are disturbances and the input $u(t)$ is located at FC_D. q_r and q_D are dependent on the current level h(t) according to Torricelli's law. The time-derivative $\dot{h}$ is given by the non-linear equation

$$\dot{h} = \frac{q_c - q_r - q_D}{A_T} \tag{7}$$

which can be linearized to a linear state space model in the form of (4) where $\Delta x(t) = \Delta y(t) = \Delta h(t)$ and

$$\begin{aligned} A &= \left(\frac{1}{A_T}\sqrt{2g}\frac{1}{2\sqrt{h_R}}(k_r d_{rR} + k_D u_R)\right) \\ B &= \left(-\frac{1}{A_T}k_D\sqrt{2g}\sqrt{h_R}\right) \\ C &= 1 \\ D &= 0 \\ E &= \left(\frac{Vf_c}{A_T} \quad -\frac{1}{A_T}k_{rR}\sqrt{2g}\sqrt{h_R}\right) \\ \sqrt{h_R} &= \frac{-Vf_c d_{cR}}{(-k_r d_{rR} - k_D u_R)\sqrt{2g}} \end{aligned} \tag{8}$$

C. Modeling the CPS

Focusing on the condenser and output tank of the phosgene separation, we first model the control systems *PC* and LC_2 from Fig. 5 excluding data and functional safety. Both span over the subsystem and component layers of abstraction. The connection between the continuous state space models of condenser and tank introduced in section V-B and the behavior of their control systems is established on the subsystem layer using hybrid automata as defined in [21].
The states V of a hybrid automaton $H = \langle X, V, E, init, inv, flow, jump, event, \Sigma\rangle$ represent control modes defining the continuous flow of real number variables X, while the transitions E specify discrete mode switches. The labeling functions *init, inv, flow* and *jump* assign predicates to each control mode v_i, where *init* assigns initial values to the variables of v_i, *inv* defines invariants for the variables, *flow* specifies the continuous change of variable values within a control mode and *jump* defines conditions for control switches. The function *event* assigns a triggering event from the set Σ of events to each control switch. In our domain model, the automata of a subprocess can exchange signals and change their control modes by the discrete event of receiving a signal and by the violation of a state invariant.
Figure 6 shows the hybrid automaton of the condenser, whose *control* state defines the control algorithm of the condenser in

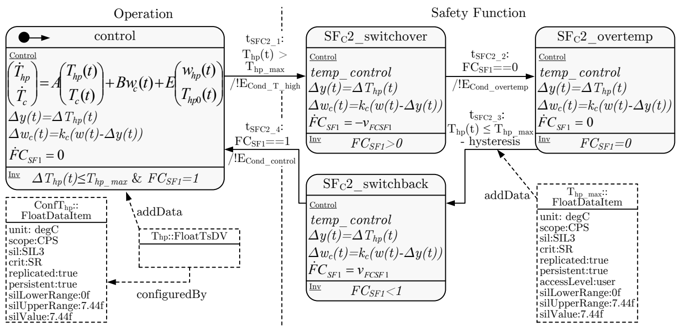

Operation
Safety Function
control
Control
$\begin{pmatrix}\dot{T}_{hp}\\ \dot{T}_c\end{pmatrix} = A\begin{pmatrix}T_{hp}(t)\\ T_c(t)\end{pmatrix} + Bw_c(t) + E\begin{pmatrix}w_{hp}(t)\\ T_{hp0}(t)\end{pmatrix}$
$\Delta y(t)=\Delta T_{hp}(t)$
$\Delta w_c(t)=k_c(w(t)-\Delta y(t))$
$\dot{FC}_{SF1} = 0$
Inv $\Delta T_{hp}(t) \leq T_{hp\_max}$ & $FC_{SF1}=1$
t_{SFC2_1}: $T_{hp}(t) > T_{hp\_max}$
/!$E_{Cond\_T\_high}$
t_{SFC2_4}: $FC_{SF1}==1$
/!$E_{Cond\_control}$
SF_C2_switchover
Control
temp_control
$\Delta y(t)=\Delta T_{hp}(t)$
$\Delta w_c(t)=k_c(w(t)-\Delta y(t))$
$\dot{FC}_{SF1} = -v_{FCSF1}$
Inv $FC_{SF1}>0$
t_{SFC2_2}: $FC_{SF1}==0$
/!$E_{Cond\_overtemp}$
SF_C2_overtemp
Control
temp_control
$\Delta y(t)=\Delta T_{hp}(t)$
$\Delta w_c(t)=k_c(w(t)-\Delta y(t))$
$\dot{FC}_{SF1} = 0$
Inv $FC_{SF1}=0$
t_{SFC2_3}: $T_{hp}(t) \leq T_{hp\_max}$ - hysteresis
SF_C2_switchback
Control
temp_control
$\Delta y(t)=\Delta T_{hp}(t)$
$\Delta w_c(t)=k_c(w(t)-\Delta y(t))$
$\dot{FC}_{SF1} = v_{FCSF1}$
Inv $FC_{SF1}<1$
$ConfT_{hp}$:: FloatDataItem
unit: degC
scope:CPS
sil:SIL3
crit:SR
replicated:true
persistent:true
silLowerRange:0f
silUpperRange:7.44f
silValue:7.44f
addData
T_{hp}::FloatTsDV
configuredBy
addData
T_{hp_max}:: FloatDataItem
unit: degC
scope:CPS
sil:SIL3
crit:SR
replicated:true
persistent:true
accessLevel:user
silLowerRange:0f
silUpperRange:7.44f
silValue:7.44f

Fig. 6. Behavior of the Condenser Control System on Subsystem Level

safe operation mode, where the mass flow of the coolant w_c is controlled to keep the temperature T_{hp} below T_{hp_max}. The control system in our case study uses a proportional controller $\Delta w_c(t) = k_c(w(t) - \Delta y(t))$ which multiplies the comparison of reference variable $w(t)$ and output $\Delta y(t)$ by a factor k_c. The invariants of the control state define that during regular control, the HCl output valve FC_{FS1} is constantly open and the temperature ΔT_{hp} is always below the threshold T_{hp_max} of 7.44 degrees Celsius. Similar to the condenser, the behavior of the output tank on subsystem level is defined as hybrid automaton shown in Fig. 7. During regular operation, the automaton controls the level of the output tank via the valve FC_D, as defined in (8).
In our domain model, the component layer as lowest layer of abstraction specifies the implementation of the control algorithms defined on the subsystem layer. Fig. 8 shows the deployment structure of the condenser and output tank. The condenser consists of five nodes which are containers for groups of hardware units connected by communication links and typically contained in a separate housing. Three of the nodes are vortex sensors which count the vortexes in the Kármán vortex street behind a bluff body in the pipe to measure the flow speed. The structural model of a vortex node in our domain model consists of a sensor (e.g. piezo elements or a physical switch) for vortex counting, pressure and temperature sensors for mass flow calculation, a vortex algorithm software calculating the mass flow from the vortex frequency *f* as well as software components for inter-node data management and communication. *Condenser_Vortex1* and *Condenser_Vortex2* measure the mass flow w_{hp} and the temperature T_{hp0}, while *Condenser_Vortex3* measures the incoming mass flow w_c and temperature T_{c0} of the coolant. The node *Condenser_Temp1* contains two temperature sensors measuring the temperature T_{hp} of the distillate. In addition to the three sensor nodes, the converter includes a node with two microcontroller units (*MCUs*) running the software components for the temperature control algorithm and the actual valve control. The converter controls the valve nodes *FCC*, *FCSF1* and *FCSF2*.
The output tank in our case study consists of three nodes: two radar level sensor nodes *Tank_Radar1* and *Tank_Radar2* as well as the *Tank_converter* node which runs the level control algorithm and controls the valve FC_D. The nodes of the converter and the tank are connected using a bus system while the connections between valves and control nodes are realized via point to point links.
On the component level the behavior of each control mode from the hybrid automaton on the subsystem level is modeled

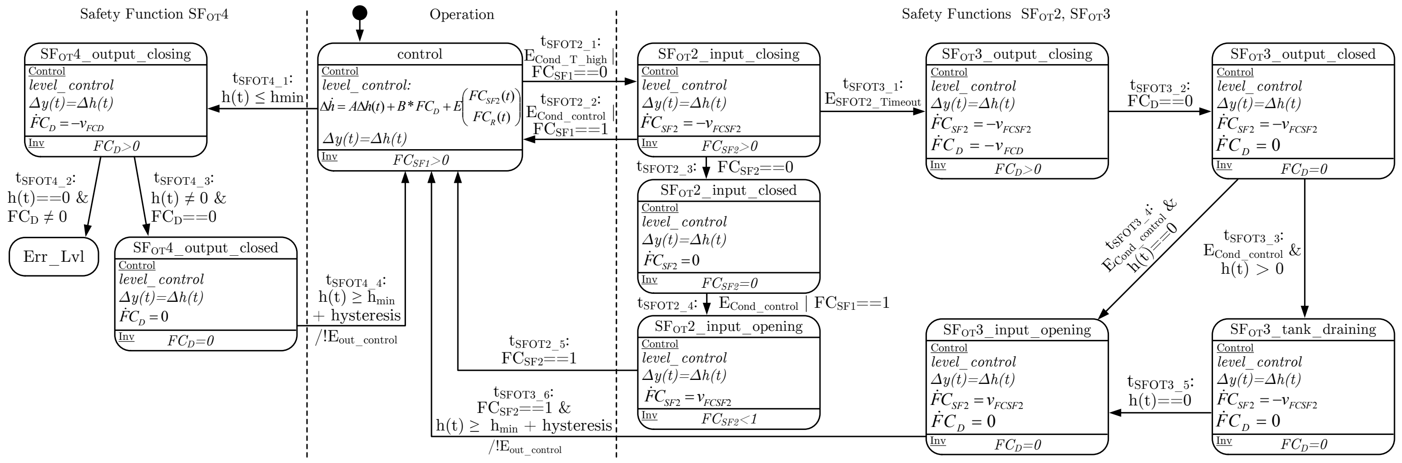

Safety Function $SF_{OT}4$
Operation
Safety Functions $SF_{OT}2$, $SF_{OT}3$
$SF_{OT}4$_output_closing
control
$SF_{OT}2$_input_closing
$SF_{OT}3$_output_closing
$SF_{OT}3$_output_closed
$SF_{OT}2$_input_closed
$SF_{OT}2$_input_opening
$SF_{OT}4$_output_closed
$SF_{OT}3$_input_opening
$SF_{OT}3$_tank_draining
Err_Lvl
t_{SFOT4_1}: h(t) ≤ hmin
t_{SFOT4_2}: h(t)==0 & $FC_D \neq 0$
t_{SFOT4_3}: h(t) ≠ 0 & FC_D==0
t_{SFOT4_4}: h(t) ≥ h_{min} + hysteresis /!$E_{out\_control}$
t_{SFOT2_1}: $E_{Cond\_T\_high}$ | FC_{SF1}==0
t_{SFOT2_2}: $E_{Cond\_control}$ | FC_{SF1}==1
t_{SFOT2_3}: FC_{SF2}==0
t_{SFOT2_4}: $E_{Cond\_control}$ | FC_{SF1}==1
t_{SFOT2_5}: FC_{SF2}==1
t_{SFOT3_1}: $E_{SFOT2\_Timeout}$
t_{SFOT3_2}: FC_D==0
t_{SFOT3_3}: $E_{Cond\_control}$ & h(t) > 0
t_{SFOT3_4}: $E_{Cond\_control}$ & h(t)==0
t_{SFOT3_5}: h(t)==0
t_{SFOT3_6}: FC_{SF2}==1 & h(t) ≥ h_{min} + hysteresis /!$E_{out\_control}$
Control
level_control
level_control: $\Delta\dot{h} = A\Delta h(t) + B * FC_D + E\begin{pmatrix} FC_{SF2}(t) \\ FC_g(t) \end{pmatrix}$
$\Delta y(t)=\Delta h(t)$
$\dot{FC}_D = -v_{FCD}$
$\dot{FC}_D = 0$
$\dot{FC}_{SF2} = -v_{FCSF2}$
$\dot{FC}_{SF2} = 0$
$\dot{FC}_{SF2} = v_{FCSF2}$
Inv
$FC_D>0$
$FC_D=0$
$FC_{SF1}>0$
$FC_{SF2}>0$
$FC_{SF2}=0$
$FC_{SF2}<1$

Fig. 7. Behavior of the Output Tank Control System on Subsystem Level

as activity diagram defining the control and data flows between the control system entities in this specific mode. Each of the activities carried out by an entity can additionally be detailed by a distinct activity diagram. Fig. 9 partly shows the activity diagram for the control mode of the condenser. The grey actions mark the standard behavior of the control system while the dotted flows and white activities are weaved from the safety aspect as described in section V-D. The mass flow w_{hp1} is calculated from the vortex frequency f_{hp1} and the temperature T_{hp0_1} and transferred via the bus using the action *transferSR* denoting safety-critical communication. Together with the current temperature value T_{hp}, the temperature control calculates a control input $controlFC_C$ which is adjusted in the *ValveControl* module of the valve FC_C.

D. Modeling Safety Aspects

Complementing the system specification, we use safety aspect models to define the safety functions of condenser and tank on the subsystem layer of abstraction as well as their realization on the component layer. To demonstrate the specification of hardware failures and their tracing to detection methods, we apply the safety concept of 2-channel redundancy by structural and behavioral weaving to the condenser domain model.

The safety functions of condenser and output tank are closely coupled. SF_C1 in the condenser ensures that the HCl which is used in other processes or even sold to contractors is not contaminated with phosgene. To prevent this, PC continuously controls pressure and temperature T_{hp} by manipulating the incoming mass flow w_c of the coolant. A demand on the safety function SF_C2 occurs when T_{hp} rises above 7.44 degrees Celsius due to insufficient cooling. In this case, the safety function assumes that phosgene has not been completely liquefied and is flowing as gas towards the valve FC_{SF1}. SF_C2 must close the valve before phosgene exits. The resulting effect is a mixture of HCL and phosgene leaving the condenser towards the output tank.

The output tank has four distinct safety functions. $SF_{OT}1$ continuously controls the tank level via the valve FC_D. $SF_{OT}2$ reacts on the flow of HCl towards the output tank evoked by the triggering of SF_C2 by closing the valve FC_{SF2} before HCl can enter the tank, directing the complete

TABLE I. SAFETY FUNCTIONS OF CONDENSER AND OUTPUT TANK

	Process Step	Mode	Control	Description
SF_C1	condenser	continuous	PC	temperature control
SF_C2	condenser	on demand	PC	prevent phosgene output
$SF_{OT}1$	tank	continuous	LC_2	level control
$SF_{OT}2$	tank	on demand	LC_2, PC	prevent HCl output
$SF_{OT}3$	tank	on demand	LC_2	prevent HCl output
$SF_{OT}4$	tank	on demand	LC_2	prevent empty pipe

condenser output back to the distillation column as reflux. $SF_{OT}3$ closes the input and output valves FC_{SF2} and FC_D in case $SF_{OT}2$ is not successful in preventing the entry of HCl into the output tank. $SF_{OT}3$ is responsible for emptying the tank via the reflux to ensure that no HCl is put out to following subprocesses. A demand on $SF_{OT}4$ is triggered when the tank level falls below a minimum threshold. $SF_{OT}4$ is responsible for ensuring that the output valve is never opened when the tank is completely empty to prevent damage to following subprocesses caused by empty pipes. Table I lists the safety functions covered by our case study.

As introduced in section III, our domain model extends the behavioral specification of the system models using the weaving functions addBehavior and removeBehavior. Fig. 6 shows the behavioral extension of the condenser automaton by the behavior of the safety functions from Table I. The safety function SF_C1 is continuously executed during safe operation in the control state of the automaton without altering the behavior. SF_C2 is executed on demand, i.e. when a dangerous event occurs that may lead to process risks which have to be mitigated. We use the weaving function *addBehavior* to add three additional states to the automaton that are executed by SF_C2. The demand for SF_C2 is modeled by the transition t_{SFC2_1} which transfers the automaton to the state *switchover* when the temperature $T_{hp}(t)$ rises above T_{hp_max} and broadcasts the signal $E_{Cond\_T\_high}$. In this state, SF_C2 closes the valve FC_{SF1} at maximum speed v_{FCSF1} which is denoted by the flow equation $\dot{FC}_{SF1} = -v_{FCSF1}$. When the valve is completely closed, SF_C2 enters the state *overtemp* in which the valve FC_{SF1} remains closed ($\dot{FC}_{SF1} = 0$) so that gaseous phosgene cannot be released and the total output of the condenser flows towards the tank. When the temperature $T_{hp}(t)$ falls below T_{hp_max} including a hysteresis preventing

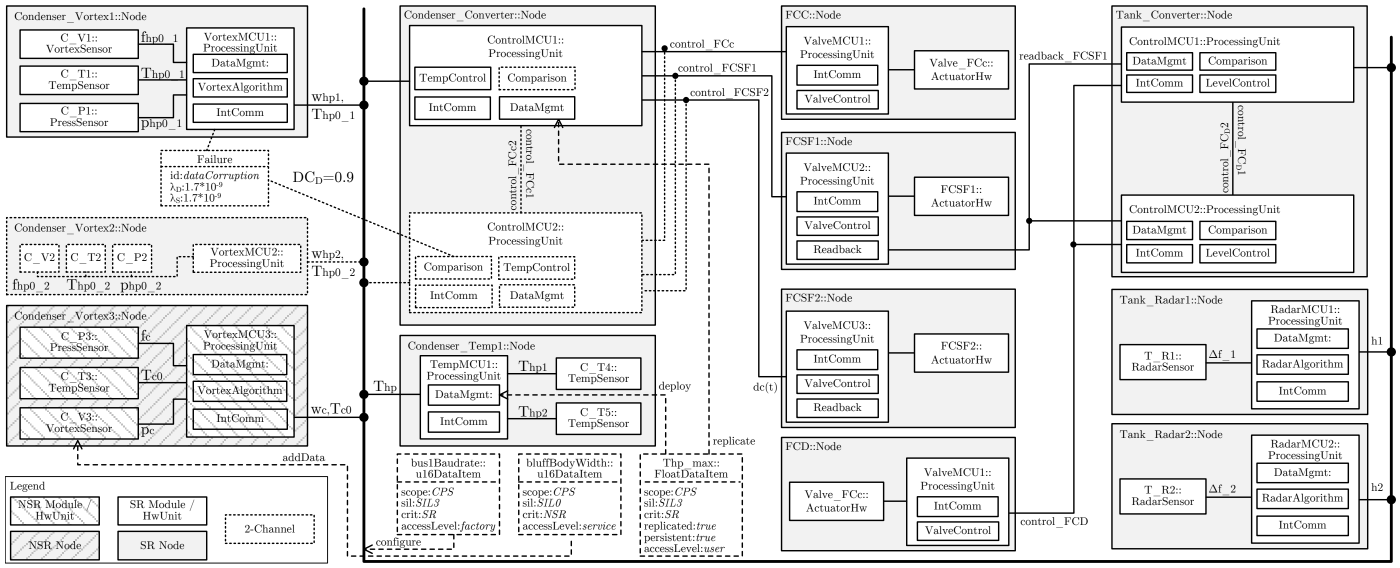


Fig. 8. Deployment Diagram of the Condenser Control System

the rapid switching of control modes, transition t_{SFC2_3} fires and the valve FC_{SF1} is opened again in the state *switchback* to finally return to the behavior of the control state when the valve is completely opened.

The same mechanism is used to attach the safety functions of the output tank to the hybrid automaton in Fig. 7. $SF_{OT}1$ is continuously processed in the control state of the hybrid automaton. The safety functions $SF_{OT}2$ and $SF_{OT}3$ are chained. $SF_{OT}2$ is entered when the valve FC_{SF1} is completely closed or the signal $E_{Cond\_T\_high}$ is received which announces the triggering of SF_C1. In both cases, the tank control closes the input valve ($\dot{FC}_{SF2} = -v_{FCSF2}$) to prevent the inflow of HCl. The safety time for $SF_{OT}2$ is defined by the time the HCl needs to flow from the condenser to the valve FC_{SF2}. If FC_{SF2} closes before the safety time exceeds, $SF_{OT}2$ returns to the control state via t_{SFOT2_3}. If the closing of the valve requires more time (denoted by the signal $E_{SFOT2\_Timeout}$), HCl enters the output tank and the transition t_{SFOT3_1} activates safety function $SF_{OT}3$ as second part of the chain. $SF_{OT}3$ closes the output valve FC_D by setting the flow $\dot{FC}_D = -v_{FCD}$ in the state *output_closing*. Once FC_D is completely closed, t_{SFOT3_2} fires and the tank level decreases due to the reflux to the column. The safety function waits for the signal $E_{Cond\_control}$ which announces regular condenser operation with complete separation of HCl and phosgene. If at this point in time the tank still contains HCl and phosgene, $SF_{OT}3$ enters the state *tank_draining* and waits for $h(t)$ to decrease to zero. If $h(t)$ equals zero, the safety function reopens FC_{SF2} in the state *input_opening*, filling the output tank with phosgene, and returns to normal operation via t_{SFOT3_6} when the level rises above a minimum threshold $h_{min} + hysteresis$.

The safety function $SF_{OT}4$ ensures that the output valve FC_D is closed when the tank level $h(t)$ decreases to zero to prevent damage to following subprocesses. The demand for the safety function is a decrease of $h(t)$ below a configurable threshold h_{min} which causes firing of the transition t_{SFOT4_1}. In the state *output_closing*, $SF_{OT}4$ closes the valve FC_D. If FC_D is completely closed, the safety function waits in the state *output_closed* for the tank level to rise over the minimum threshold with hysteresis and switches back to normal operation via t_{SFOT4_4}. If however the tank level decreases to zero before FC_D is completely closed, a safety-critical error *Err_Lvl* occurs representing the failure of $SF_{OT}4$ in preserving a safe system state for the process step.

The safety functions of condenser and output tank have to react by transferring their process step into a safe state during the interval between the occurrence and the consequence of dangerous events referred to as *process safety time*. Fig. 10 shows the safety automaton of condenser and output tank. As described in section III, in addition to the modes of operation the safety automaton defines the process safety times and safety-critical errors for each safety function. For SF_C2, the automaton defines the safety time $tsafe_{SFC2}$ whose violation leads to the critical error Err_FC_{SF1}, and the chained safety times $tsafe_{SFOT2}$ and $tsafe_{SFOT3}$ leading to the error state Err_FC_D. Section V-F argues that the occurrence of these safety-critical errors depends on the configuration on both subsystem and component layer and uses the automaton in Fig. 10 to define a reachability problem supporting the prove of correct system configurations regarding functional safety.

To support reuse and reduce modeling complexity, the safety aspect defines safety concepts which can be used to implement safety functions. These safety concepts are weaved into both structural and behavioral models on the component layer. Fig. 11 shows the structural aspect model of the safety concept "2-channel" which introduces a system of two sensors, two processing units and two actuators. The sensors send a dynamic data item to both processing units where they are compared by software modules. The comparison modules on each processing unit then calculate a derived data item and exchange it to verify the calculation on each MCU. Each of the processing units controls an actuator connected to the same physical process. This safety concept introduces redundancy to safety-related control tasks.

Fig. 8 shows the weaving of the 2-channel safety concept to the condenser control structure. Both the node *Condenser_Vortex2* and the processing units *VortexMCU2* and *ControlMCU2* and

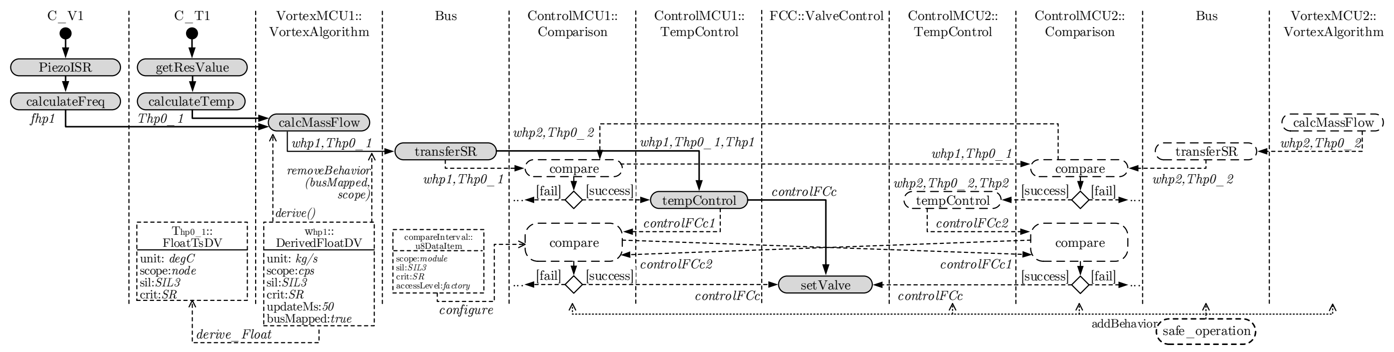


Fig. 9. Weaving of 2-Channel Concept into Condenser Behavior

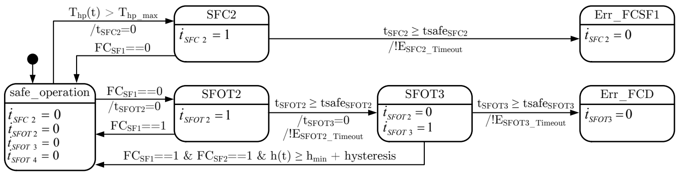


Fig. 10. Safety Automaton of Condenser and Output Tank

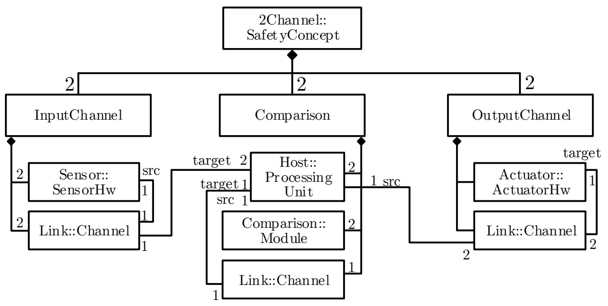


Fig. 11. Structural Aspect Model for Safety Concept ”2-Channel”

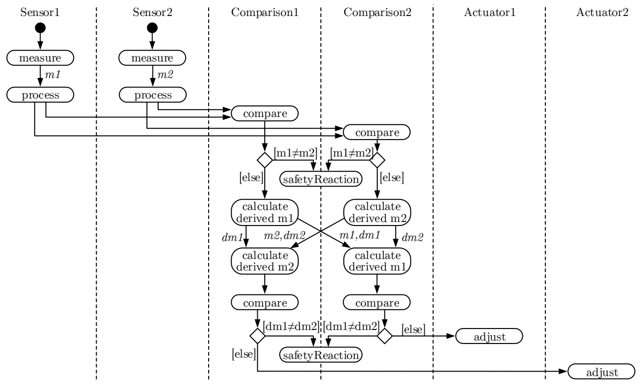


Fig. 12. Behavioral Aspect Model for Safety Concept ”2-Channel”

the corresponding bus channels are part of the 2-channel safety concept added to the model via the weaving function *addEntity*. Well formedness rules ensure the correct binding of aspect and domain models using the types of the elements in the aspect model as reference.

The hardware failure information introduced in section III can be used on the component level to document the identified failure modes from a safety analysis and trace their implemented countermeasures to support system certification. In Fig. 8 this is illustrated using the failure *dataCorruption* which occurs at the memory of VortexMCU1 and leads to a dangerous failure with the rate $\lambda_D = 1.7 * 10^{-9}$. The 2-channel safety concept acts as diagnostic measure against the dataCorruption error with a diagnostic coverage DC_D of 90% regarding dangerous failures.

Complementing the structure, a behavioral description of safety concepts is part of the aspect models as shown for the 2-channel concept in Fig. 12. The activity diagram defines the cross-verification data exchange between the concept entities. An exemplary weaving of the safety concept is shown in the activity diagram of the condenser in Fig. 9 using the weaving function *addBehavior* to add entities, activities, control and data flows to the original behavior with correspondence to the safety concept ensured by well-formedness rules. The aspect-oriented approach improves the accessibility of common safety patterns like the 2-channel approach to domain experts without explicit safety knowledge and encourages extended reuse from implementation to documentation and certification artifacts when reusing complete safety concepts.

E. Modeling Data Aspects

We use the data aspect to add configurability to condenser and tank on subsystem and component level and demonstrate how a specific configuration can be defined and bound to the domain models. On the subsystem level, the aspect focuses on the parameters of process steps and their safety functions, providing a process owner’s point of view. The major part of the data represent customization and data processing of the CPS parts realizing the process control on the component level. Similar to the safety aspect, weaving functions connect device variables and static data items with the behavioral and structural models on both layers, where the weaving on component level corresponds to the customization of a mass market device to a specific application.

The hybrid automata on the subsystem layer use device variables in their flow equations and static data items in transition guards and invariants. The weaving function *addData* connects the device variables and static data items from the data aspect to the variables and constants of the hybrid automata. Fig. 6 shows the connection of the device variable T_{hp} and its configuration data item $ConfT_{hp}$ as well as the maximum value T_{hp_max} to the condenser automaton. This reduced data set shows that inline modeling of all configuration data values would magnify the complexity of behavioral models beyond the point of efficient handling. Using our aspect-oriented approach, we promote to hold complete data models externally, e.g. in a separate database and trace the connections to the variables in the automata via the aspect weaving.

On the component layer, the origin of the device variables introduced in the hybrid automata is defined by connecting the device variables to producing software modules in the structural and deployment viewpoint using the weaving function *producedBy*. Dependent on their scope, the device variables are then distributed to all data management modules in the scope of the device variable. Fig. 8 illustrates the source modules of device variables by the labels of their outgoing links. Hardware-related characteristics used by software modules are bound to the respective hardware unit or link via the *addData* weaving function, configuration of software modules is done using the function *configure*. In Fig. 8, the construction variable bluff body width of the vortex sensor C_V3 is defined using the static data item *bluffBodyWidth* and the baudrate of the bus system connecting the nodes is set using the data item *bus1Baudrate*. The storage and replication of static data items is defined in the structural models of the component by the weaving functions *replicate* and *deploy*, as shown in Fig. 8 for the example of the maximum temperature at the condenser T_{hp_max}, which is deployed at *TempMCU1* of the condenser and replicated to *ControlMCU1*.

Configuration data influences the behavior of the control system on the component layer by manipulating the control flows of the overview and detailed activity diagrams. The scope and distribution settings of device variables can add sending behavior via the *addBehavior* weaving function as shown in Fig. 9 for w_{hp1}. The common case of behavioral manipulation is the binding of activities and control paths to device variables as shown for the control state of the tank subsystem in Fig. 13.

F. Model-based Verification

During our case study, we focused on inconsistencies arising from the interdependence between the influences of data and safety aspects on the domain model. The configuration of a domain model via the data aspect can lead to an unsafe system due to structural and behavioral configuration errors.

Structural inconsistencies are static misconfigurations which prevent correct system execution, arising from the weaving of the data aspect into domain models with varying complexity. They can be mitigated using the well-formedness rules connected to the weaving functions of our aspect models. For example in the hardware and deployment viewpoint, the safety aspect adds wfrs that constrain the deployment of elements to safety considerations:

$$\boldsymbol{Wfr_{sil\_deploy}} : \forall dep_i \in Deployee, dc_i \in DeployContext : \\ (dep_i, dc_i) \in deploy \rightarrow sil(dep_i) \leq sil(dc_i) \quad (9)$$

wfr_{sil_deploy} ensures that elements dep_i may only be deployed on hardware or partitions dc_i that have a SIL high enough to provide the safety demanded by dep_i. Additional wfrs restrict the control of safety-related actuator and sensor hardware. In our case study, the deployment of device variables to producing modules in Fig. 8 must be checked using wfrs that guarantee that the producing software modules of sr device variables are sr themselves, have a sil equal or higher than those of the device variable and are deployed on an appropriate hardware unit.

Apart from these errors within the scope of a single viewpoint, more complex scenarios arise when interactions of multiple system parts defined on multiple views must be considered. The device variable w_{hp1} e.g. is produced at the vortex sensor *Condenser_Vortex1*, and transferred to both Control MCUs of the condenser where it is compared before further processing (see Fig. 8 and 9). If one of the compared w_{hp} values is configured with a narrow scope, it is not transferred and thus not usable by the comparison module. In addition, static inconsistencies can arise due to influence of both data and safety aspects on the same domain model elements. During our case study, this conflict occurred in combination with check and update dependencies of data items. Check dependencies ensure that the configuration of a data item d_1 is only accepted by the system if the attached data items $d_2..d_n$ perform checks including range compliance on their values. If the safety aspect however removes parts of an activity diagram by the weaving function *removeBehavior* to which the dependent data item $d_{i>1}$ is attached, data item d_1 cannot satisfy its check dependencies and thus cannot be configured. In industrial scale CPS with multiple processes and control loops, even the basic single-view cases discussed above become critical and the set of possible inconsistencies are not manually controllable. Our case study therefore confirms the need for algorithmic support to concisely monitor the structural consistency of safety and data aspects with respect to the domain model during system development and operation.

Inconsistencies arising due to the influence of data items on the system behavior are more difficult to detect. As illustrated in Fig. 10, safety functions of a control system must mitigate risks within the process safety time defined in the hybrid automata on the subsystem layer from the process owner's point of view. In contrast to this, device vendors deliver systems which are highly configurable via data items manipulating structure and behavior on the component layer. As motivated before, this raises the problem of verifying that a system configuration given by an instanced data model does not prevent the safety functions from transferring the system into a safe state in case of a dangerous event within the safety time. A longer query interval of the valve value FC_{SF1} due to energy restrictions e.g. leads to longer activation and thus shorter available reaction time of the safety function $SF_{OT}2$.

To detect and prevent these error scenarios, we propose to extend the activity diagrams on the component layer by worst case execution times (WCET) determined for each action and accumulated for activities in the overview activity diagrams. Since every state in the hybrid automaton of a control system is represented by a distinct overview activity diagram at the component layer, the accumulated execution times of a diagram iteration can be interpreted as the WCET of a transition exiting the corresponding state in the hybrid automaton of the control system. Through the manipulation of control flows in activities, the influence of static data items and device variables on the WCET becomes traceable and can be automatically evaluated from the activity diagrams, specifying the correlation between data and execution time.

Fig. 13 shows excerpts from the tank and safety automaton and the overview activity diagram of the tank *control* state. Transition t_{SFOT2_1} marks the activation of safety function

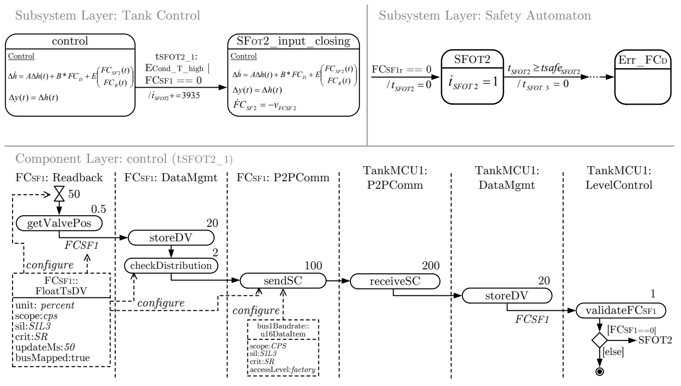


Fig. 13. Dynamic Verification

$SF_{OT}2$ by detection of the closing of valve FC_{SF1}. The example activity diagram shows that the configuration of the device variable FC_{SF1} heavily influences the execution time of the transition: *updateMs* defines the update interval of 50ms, *busMapped* defines the transfer function used and the static data item *bus1Baudrate* influences the transfer time of the device variable. In the example, the data-including WCET of the transition t_{SFOT2_1} is 393.5ms. This delay is added to the counter t_{SFOT2} in the safety automaton which models the occurrence of a safety-critical error in case of a safety timeout. Note that the timer in the safety automation starts at the condition $FC_{SF1r} == 0$ which references the real value of the valve independent of control system delays.
Using this approach, the ability of a control system to perform its safety functions within the safety time under a given configuration can be expressed as reachability problem, where the error states of the safety automaton may not be reached by the configuration under test. When moving from our simple example to more complex models, e.g. the 2-channel safety scheme weaved into the condenser behavior in Fig. 9, the need for automated verification becomes obvious. Such a verification approach must prove that the error states of the system, e.g. Err_FC_D in Fig. 13, can never be reached given the influences of the data configuration on the execution times of the hybrid automata. Existing approaches use over-approximation of the continuous flows inside the automata states to manage termination and execution time and space. We currently investigate the application of the tool *SpaceEx* which was used by Frehse et. al. in [22] for control system verification to our data-driven approach.

VI. Conclusion and Future Work

In this paper, we extend our domain model for hierarchical modeling of CPS in industrial automation introduced in [6] by defining comprehensive aspect models for the cross-cutting concerns of functional safety and data. Both aspects are key concerns in process automation systems. We use the concepts of weaving functions and well-formedness rules to give a concise definition of the valid connections between aspect and domain models, providing a baseline for isolated formal reasoning about data and safety. We apply our aspect models to a case study on the development of an industrial CPS in MDI production, focusing on the conflicts between functional safety and configurability. We identify possible consistency conflicts due to the interdependence between safety and data aspects and the domain model. Based on our case study, we propose approaches for static and dynamic verification of the safety of given configurations and deduce the need for algorithmic support for monitoring and configuring large-scale industrial safety-critical CPS.
Future work includes the formal definition of configuration and data conflicts in industrial CPS and research on algorithms enabling offline and online monitoring of configurations conflicting with the functional safety of the CPS.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," University of California, Berkeley, USA, Tech. Rep. UCB/EECS-2008-8, Jan 2008.

[2] P. Derler, E. Lee *et al.*, "Modeling cyber-physical systems," *Proc. IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[3] R. Rajkumar, I. Lee *et al.*, "Cyber-physical systems: The next computing revolution," in *Proc. DAC 2010*, Jun 2010, pp. 731–736.

[4] A. Colombo, T. Bangemann *et al.*, *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*. Springer, 2014.

[5] *Functional safety of electrical/electronic/programmable electronic safety-related systems*, IEC Std. 61 508, 2010.

[6] D. Kuschnerus, A. Bilgic *et al.*, "A hierarchical domain model for safety-critical cyber-physical systems in process automation," in *Proc. INDIN 2015*, Cambridge, UK, Jul. 2015.

[7] K. Petersen, R. Feldt *et al.*, "Systematic mapping studies in software engineering," in *Proc. EASE'08*, Bari, Italy, Jun 2008, pp. 68–77.

[8] S. Khaitan and J. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *IEEE Systems Journal*.

[9] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, UK, Tech. Rep. EBSE-2007-01, Jul 2007.

[10] B. Genge and C. Siaterlis, "Physical process resilience-aware network design for SCADA systems," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 142–157, 2014.

[11] N. Saeedloei and G. Gupta, "A logic-based modeling and verification of CPS," *SIGBED Rev.*, vol. 8, no. 2, pp. 31–34, 2011.

[12] C. Schwarz, "Modelling a real-time control system using parameterized linear hybrid automata," in *Informatik 2011*, Bonn, Germany, Oct 2011, pp. 328–328.

[13] L. Besnard, A. Bouakaz *et al.*, "Timed behavioural modelling and affine scheduling of embedded software architectures in the AADL using polychrony," *Sci. Comp. Prog.*, 2014.

[14] A. Rajhans, A. Bhave *et al.*, "Supporting heterogeneity in cyber-physical systems architectures," *IEEE Trans. Autom. Control*, vol. 59, no. 12, pp. 3178 – 3193, 2014.

[15] F. Saunders, J. Rife *et al.*, "Information flow diagram analysis of a model cyber-physical system: Conflict detection and resolution for airport surface traffic," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 28, no. 12, pp. 26–35, 2013.

[16] T. Stauner, "Systematic development of hybrid systems," Ph.D. dissertation, Technische Universitaet Muenchen, 2001.

[17] M. Zheng, J. Sun *et al.*, "Towards a model checker for NesC and wireless sensor networks," in *LNCS*. Berlin Heidelberg: Springer-Verlag, 2011, vol. 6991.

[18] R. France, I. Ray *et al.*, "Aspect-oriented approach to early design modelling," *Softw., IEE Proc.*, vol. 151, no. 4, pp. 173–185, Aug 2004.

[19] H. Pirkl, J. Bolton *et al.*, "Process for the preparation of highly pure 2,4'-methylenediphenyldiisocyanate," Patent EP1 561 746A2, Jan, 2005.

[20] R. Shah, A. Alleyne *et al.*, "Dynamic modeling and control of single and multi-evaporator subcritical vapor compression systems," Air Conditioning and Refrigeration Center. College of Engineering. University of Illinois, Tech. Rep. ACRC-TR-216, Aug. 2003.

[21] T. Henzinger, "The theory of hybrid automata," in *Ver. Digital Hybrid Systems*. Berlin Heidelberg: Springer-Verlag, 2000, vol. 170.

[22] G. Frehse, A. Hamann *et al.*, "Formal analysis of timing effects on closed-loop properties of control software," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*, Dec 2014, pp. 53–62.

Efficient Identification of Safety Goals in the Automotive E/E Domain

Rolf Johansson

Abstract— **This paper addresses the problem of how to identify all safety goals for an item in the automotive E/E domain. The paper gives a background on the problem of hazard analysis and risk assessment in general, and for the automotive domain in particular. A key factor for success is to identify all the relevant hazardous events, which task constitutes a paradox. Either the specification of the possible driving situations and the system hazards are done too general and abstract implying a too conservative analysis, or done too detailed and specific ending up with an almost infinite list of hazardous events to consider. This paper addresses this paradox by the formulation of a number of rules enabling to reduce the potentially infinite set of candidates of hazardous events to a limited number, still sufficient to cover all safety goals. Besides that it enables solving the paradox of becoming both detailed and limited, it also can be used as a tool for reviewing the completeness of a set of safety goals.**

Index Terms— **Hazard analysis, Automotive, ISO 26262.**

I. Introduction

In all domains of functional safety, it is essential to perform a complete and correct Hazard analysis and risk assessment (HA&RA). The purpose of this activity is to identify and categorize all Hazards, i.e. all potential sources of accidents caused by erroneous behaviour of the function under consideration. In the automotive domain, the rules for HA&RA are given by part three of the ISO 26262 standard [1]. The goal of this HA&RA is to produce the so called Safety Goals. The rest of the standard prescribes how to guarantee that all these Safety Goals in the end are fulfilled, thus implying functionally safe road vehicles. However, in order for the complete vehicle to act functionally safe, first the set of items analysed must be complete, and secondly, for every item the set of Safety Goals must be complete. This paper addresses the latter problem.

In the process to identify the Safety Goals, the standard ISO 26262 prescribes to analyse all Hazardous Events that might have an impact on this set. This means that all possible failures of the Item of concern should be considered, and for all driving scenarios and all environmental conditions. The combined effect of the driving scenarios and the environmental conditions are called ‘situation’ in the ISO 26262 terminology. As there is no predefined standard set of situations, this would potentially generate an infinite number of Hazardous Events to analyse. The question is whether we can come up with a way to identify if there is a limited set of Hazardous Events that completely identifies the full set of Safety Goals. The reason why this would work is that a large number of Hazardous Events do not contribute to the identification of a unique Safety Goal. What we need is a way to identify a limited set of Hazardous Events, which still cover all Safety Goals for the item.

This work has been financed by The Swedish government agency for innovation systems (VINNOVA) in the FUSE project (ref 2013-02650).

Rolf Johansson is with SP Technical Research Institute of Sweden (e-mail: rolf.johansson@sp.se).

In ISO 26262 it is prescribed that the HA&RA shall be done, and that there shall be performed a verification activity showing the “completeness with regard to situations and Hazards”. However, there is no information how to solve the problem of showing completeness. This paper addresses this problem. The remainder of the paper is structured as follows. The next section presents the HA&RA activity of ISO 26262, and why it is considered as a hard problem to identify a limited, but still efficient set of Safety Goals. In section III is introduced the concept of formulating rules to identify which Hazardous Event that are of interest when formulating the Safety Goals. Section IV is giving a structure how to categorize all possible pairs of Hazardous Events. The following section formulates the set of rules necessary, and section VI then shows the completeness and consistency of this set. The paper ends with a summary and conclusions.

II. Background

It is well known in the area of functional safety, that the quality of the HA&RA is critical for the relevance of all other risk reducing activities prescribed by a standard. In Birch et al. [2] is discussed how the complete safety case is dependent on a proper identification of the Safety Goals in the automotive domain, which in turn is dependent on the Hazardous Events. There are a number of techniques in this area which have in common that they address how to avoid missing any candidate Hazard. Often mentioned are Preliminary Hazard Analysis, HAZOP and FMEA. Other recommendations exist like a generic method by Jesty et al. [3] based on a state machine model of the transitions between a failure occurring in a system and a Hazardous Event.

In the following we are addressing specifically the HA&RA as specified for the automotive domain. Even though there are many similarities, there are also a number of fundamental differences between the industrial domains, as is pointed out in a comparative study by Blanquart et al. [4]. Section 7 of [1] prescribes in detail how an HA&RA shall be done according to ISO 26262. In principle it consists of the following activities:

- Identify all relevant Situations
- Identify all relevant Hazards
- Combine Situations and Hazards to Hazardous Events
- Perform classification of Hazardous Events
- Identify Safety Goals covering all Hazardous Events
- Verify completeness and consistency

The end result of the HA&RA is a set of Safety Goals, each having an ASIL attribute that is limiting the occurrence of a certain Hazard. One Safety Goal may cover several different Hazardous Events, which implies that it gets the highest ASIL value among those. For each Hazardous Event, the ASIL attribute is calculated by determining a factor for each of: severity (S), exposure (E) and controllability (C). For each given Hazardous Event these factors may be determined by application experts, and may include driver controllability experiments, field data collection, etc. The problem of how to determine ASIL attribute of a given Hazardous Event is not addressed in this paper. Here is rather the focus how to find a list of Hazardous Events for which it is worth getting high confidence in the E, S and C factors.

As said above, in the automotive E/E domain, the HA&RA problem is explicitly decomposed into finding the Situations and the Hazards, and analysing all resulting effects. When performing this work, different organizations have different templates for identification of the Hazardous Events of concern. A frequent pattern is to separate the driving conditions from the environmental conditions. The former is focused on the state and the intended manoeuvres of the vehicle (driving at 70km/h, full braking, etc), and the latter describes the state outside the vehicle (dark, wet road, playing child on the road etc). There are initiatives to formalize the potentially infinite number of Situations. In Jang et al. [5] the authors present a template where they decompose the Situation into properties for vehicle, road and environment. For each of these they propose a number of properties to determine, each with some standard alternatives. Even in this rather simplistic template, the number of possible alternatives for each situation is about 50 millions. Then are still not the different possible Hazards considered. Multiplying the number of possible Hazards for a given Item, with 50 million would generate a prohibitively long list of possible Hazardous Events. This is not what the authors propose, but this shows that even if there is a standardized set of Situations and/or of Hazards, there will be a need for techniques to identify which Hazardous Events that are important for identification of the Safety Goals.

The German automotive organization VDA has created a standardized list of situations [6]. The aim of this list is to harmonize the determination of the exposure factor between vehicle OEMs when performing the HA&RA.

Martin et al [7] presents a study where the original list of Hazardous Events consists of 640 candidates. This list was reduced by checking the ‘plausibility of the combinations’ into 121 Hazardous Events. After this reduction, they started the classification and ASIL determination. This presented example is far from unique in the number of Hazardous Events to consider, and still it might be the case that this HA&RA is not detailed enough to allow a precise (not too conservative) formulation of Safety Goals. This is why this paper addresses the task of enabling of automatic reduction of a potentially infinite list of Hazardous Events.

As pointed out in [8] it becomes even more important to find a carefully chosen set of Safety Goals when introducing vehicles capable of highly automatic driving (HAD) or even autonomous vehicles. The implications of many such Safety Goals are spread on a larger part of the E/E systems of the vehicle. This implies that ending the HA&RA activity with a few Safety Goals that could be regarded as too unelaborated, and thus potentially too conservative, may generate a significant increase in cost of the vehicle. This also motivates why it is important to identify rules assisting in formulating an efficient set of Safety Goals.

III. Rules for Identification of Dominance and Non-Dominance

There are a number of cases when adding a new Hazardous Event would not extend the list of already identified Safety Goals. Such Hazardous Events are of no interest, as the objective of the list of Hazardous Events, is to identify the list of Safety Goals. It would be beneficiary to have a set of rules that automatically can check whether a given candidate Hazardous Event would generate a new Safety Goal, or if it can be considered as redundant. We call such rules Dominance rules, as they identify if one Hazardous Event can be identified as dominated by other already identified Hazardous Events. Obviously we want to reduce a given list of Hazardous Events so that all the dominated ones are omitted from the final list.

In a similar way it would be efficient to have a set of rules that could clearly identify if a Hazardous Event will generate a unique Safety Goal that is not covered by any other Safety Goal. We call such rules Non-Dominance rules, as they identify Hazardous Events that cannot be identified as dominated by any other already formulated Hazardous Event.

In this paper we identify eight explicit rules that together cover all cases for determining dominance and non-dominance, respectively. These rules can be used in at least two ways. The first use case is to review a list of candidate Hazardous Events, and remove all of these that can be shown as dominated by any of the others. For the remaining Hazardous Events, it is then possible to show that they pairwise show non-dominance. The second use case is to review a list of Hazardous Events with respect to its completeness. This means that rules for non-dominance are used to identify candidates missing in the list.

IV. Categorizing Hazards and Situations

Today, different organizations have a little bit of difference in their methodology how to list the Hazardous Events and how to perform the resulting analysis. For the following

Table 1. Example extract of a Hazardous Event table

Hazardous Event ID	Situation	Hazard	Exposure	Controllability	Severity	Integrity Value
HE1	Driving (under all conditions)	Complete loss of steering functionality	E4	C3	S3	ASILD
HE2	Driving at high speed	Complete loss of steering functionality	E4	C3	S3	ASILD
HE3	Driving at high speed in heavy rain	Complete loss of steering functionality	E3	C3	S3	ASILC
HE4	Driving at high speed in heavy rain	Steering angle >20% wrong	E3	C3	S3	ASILC
HE5	Driving at high speed in heavy rain	Steering angle 5%-20% wrong	E3	C2	S3	ASILB
HE6	Driving at medium speed	Steering angle 5%-20% wrong	E4	C2	S1	ASILA
…	…	…				…

discussion, it is not necessary to include so many columns in the table that often are used.

In the following, we will use an example Item that we call Lane Keeping Assistance in Steering (LKA Steering). Once activated, this functionality takes the responsibility for the vehicle to stay in lane. Unless overridden by the driver, the LKA controls the steering of the vehicle. In the Table 1 is depicted a set of example Hazardous Events for the chosen example.

Even in this simplified example, it is not obvious if the part of the list of Hazardous Events (HE) in Table 1 is long enough to generate the identification of all Safety Goals, or if some of the HE are not contributing at all to the analysis. Let us first have a look on the effect of the different classification factors: Exposure (E), Controllability (C), and Severity (S). When setting up a detailed list of HE, it is of interest to identify the situations which constitute a border between two different values for at least one of the factors E, C or S. In our example this means that we shall identify the sizes of the steering angle failure, and the Situations, where (at least) one of the factors changes from one level to another. In the next section we look a little deeper in the question how these columns relate to each other.

A. *Analysis of Exposure, Controllability and Severity*

The Exposure factor is a direct function of the Situation, and independent of the Hazard. The definition of the E factor is that it categorizes how often a vehicle is in a given situation. If the situation is very general, the E factor will be higher than if we confine the situation. For example, comparing the HE2 and HE3 in Table 1, they only differ in how specific the situation is defined. For the more general situation of HE2 we argue that the factor should become E4, while the confinement made in HE3 implies a lowering of the E factor to E3. This lowering in turn implies a lowering of the resulting ASIL attribute. The HE2 is the more conservative case to consider. This means that it is a valid classification, but it might become too restrictive if there is no situation other than those in HE3 that will have the same effect on Controllability and Severity for the given Hazard.

The Controllability factor is a function of not only the situation, but also of the Hazard and the Severity. The interpretation of the C factor would be expressed as: “How easy is to avoid the specified Severity in this Situation given this Hazard”. In our example (comparing HE4 and HE5) we say that there is a limit when the steering angle suddenly becomes 20% wrong, for the driver being able to avoid an S3 accident, when driving at high speed in heavy rain. For an error of more than 20% we consider it a C3, while staying in the interval between 5% and 20%, it will be lowered to a C2. As before, it is not the point of this example to be fully correct, but to illustrate the principles.

The Severity factor is also a function of all: the Situation, the Hazard and the Controllability. This means that the S and C factors respectively are mutually dependent on their interpretation for a given Hazardous Event. The interpretation of the S factor would be interpreted as “What might be the Severity given the specified C-factor in this situation given this Hazard”. In our example list of Hazardous Events, there is a limit when we compare HE5 and HE6. The difference between these two cases is that the Severity is reduced when lowering the speed to medium.

When looking at all three of these factors, we conclude that Severity and Controllability are dependent and should be interpreted in any actual pair. It makes sense to interpret the Controllability factor as how easy it is to avoid a given Severity.

We conclude that when looking for the dimensioning Hazardous Events, it would be of interest to find those Situations and Hazards where any of the three factors E, C or S will change its value. The reason for this is that any such case also will imply a change in the ASIL attribute value. We conclude that there is no meaning of having two Hazardous Event candidates that only differs a little bit in the definition of the situation, if this will not imply any change of any of the E, C, or S factors.

The question is whether all changes of Situations implying a change in any of the E, C, or S factors are relevant to consider, and what set of Hazards to take into account. This question is further elaborated in the following sections.

B. Categorizing Situations

When comparing two Situations, this means that both the driving scenarios and the environmental conditions are considered. When saying that two Situations are identical, this implies identity for everything specifying a Situation. One Situation can be seen as a special case of another. In our example, the Situation of HE1 (driving, under all conditions) can be seen as a general Situation of which the Situation of HE2 (driving at high speed) is a special case. Two Situations can also be seen as mutual exclusive. The Situations 'driving at high speed' (HE2) and 'driving at medium speed' (HE7), can never include any common scenario. Finally two Situations can be over-lapping. This implies that some special Situations only may occur according to one of the Situations, some only to the other, and some to both. These four possible relations are depicted in figure 1.

.

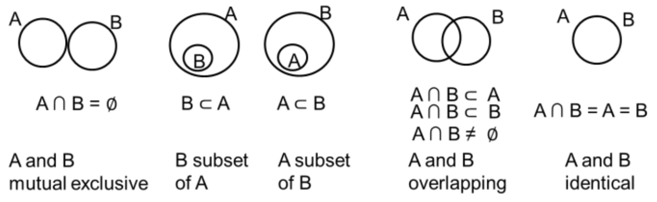


Fig. 1. Categorizing relations between two Situations A and B

To clarify the subset relation we can formulate two implications:

A⊂B ⇨Any possible situation in A will also be a possible situation in B.

A⊂B ⇨Guaranteeing the absence of any situation in B, will also guarantee the absence of that situation in A.

For our example we can list the following Situations:

A: Driving (under all conditions)
B: Driving at high speed
C: Driving in heavy rain
D: Driving at high speed in heavy rain
E: Driving at low speed on dry road

Then we can derive the following pairwise relations:

B⊂A; C⊂A; B∩C≠∅ (overlapping); D⊂B; D⊂C; D∩E=∅ (mutual exclusive)

Testing the conclusions as above, on some of these relations, we get:

D⊂C ⇨ Any Situation that may be characterized as 'Driving at high speed in heavy rain' may also be characterized as 'Driving in heavy rain'.

D⊂C ⇨ Guaranteeing the absence of any Situation that may be characterized as 'Driving in heavy rain' will imply the absence of any Situation possible to characterize as 'Driving at high speed in heavy rain'.

B⊂A ⇨ Any Situation that may be characterized as 'Driving at high speed' may also be characterized as 'Driving'.

B⊂A ⇨ Guaranteeing the absence of any Situation that may be characterized as 'Driving' will imply the absence of any Situation possible to characterize as 'Driving at high speed'.

This is in line with our intuitive understanding of these relations. The relations of these five example Situations can be depicted as either a Venn diagram or as partially ordered relations shown in the figure 2 below.

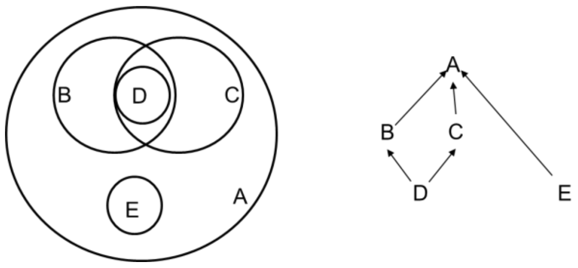


Fig. 2. Relations between example Situations

C. Categorizing Hazards

In a similar way as for the possible Situations, we can also categorize the possible relations between any two Hazards. Naming the two Hazards X and Y, respectively, the possible relations are as depicted in figure 3.

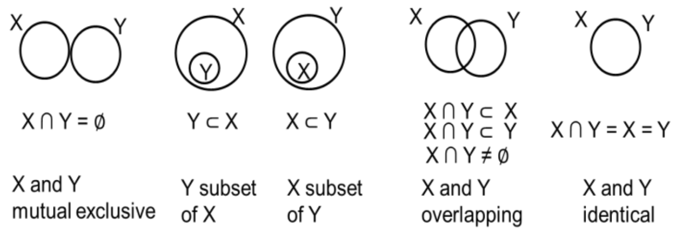


Fig. 3. Categorizing relations between two Hazards X and Y

To clarify the subset relation we can formulate two conclusions:

X⊂Y ⇨ Any possible Hazard in X will also be a possible Hazard in Y.

X⊂Y ⇨ Guaranteeing the absence of any Hazard in Y, will also guarantee the absence of that Hazard in X.

For our example we can list the following Hazards:

X: complete loss of steering functionality
Y: steering angle delayed too late >0.5 s

Z: steering angle more than 20% wrong
V: steering angle more than 5% wrong
U: steering angle between 5% and 20% wrong
W: any loss of steering functionality

Then we can derive the following pairwise relations:
X⊂Y; X⊂Z; X⊂V; X⊂W; U∩Z=∅ (mutual exclusive); Y∩Z≠∅ (overlapping); U⊂V; Z⊂V; V⊂W; Y⊂W; U⊂W; Z⊂W

Testing the conclusions as above, on some of these relations, we get:

X⊂V ⇨ Any Hazard that is characterized as 'Complete loss of steering functionality' could also be characterized as 'Steering angle more than 5% wrong'.

X⊂V ⇨ Guaranteeing the absence of any Hazard that is characterized as 'Steering angle more than 5 % wrong' will also imply the absence of any Hazard possible to characterized as 'Complete loss of steering functionality'.

Z⊂W ⇨ Any Hazard that is characterized as 'Steering angle more than 20% wrong' could also be characterized as 'any loss of steering functionality.

Z⊂W ⇨ Guaranteeing the absence of any Hazard that is characterized as 'Any loss of steering functionality' will also imply the absence of any Hazard possible to be characterized as 'Steering angle more than 20% wrong'.

If the difference between the two Hazards X and W, was not completely clear before, these clarifications have hopefully made the semantics of any of the Hazards in the list above clearer. The relations of these six example Hazards can be depicted as either a Venn diagram or as partially ordered relations shown in the figure 4 below.

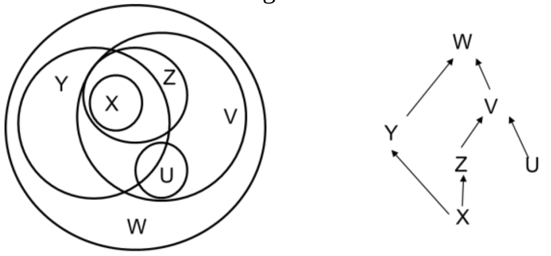


Fig. 4. Relations between example Hazards

D. Categorizing the Effects on the E, C and S Factors

In the previous sections we have categorized the relation between two Hazardous Events by first looking at the Situation column and then at the Hazard column. The remaining columns necessary to categorize are the ones for Exposure, Severity, Controllability, and the concluded ASIL value. As pointed out earlier, the important thing when comparing two HE, is the resulting ASIL value. Any difference in the E, C or S factors will imply a difference in the resulting ASIL value. The E, C, S factors are still important to list in the HE tables because they give guidance to the classification of the Situations and the Hazards. When choosing how confined/general a certain Situation or a certain Hazard should be expressed, the ideal cases would be those that result in differences in any of the E,C or S factors.
Once a Hazardous Event is formulated, for the purpose of determining dominance, it is sufficient to only consider the resulting ASIL value. The relation between ASIL values is easier to categorize than the relation between Situations or between Hazards. In fact there are three possible relations. Either the two HE have identical ASIL values, or the one is higher or the other is higher. The symbols =, > and < are used for this in the following sections of this paper.

V. Rules for Reducing Candidates of Hazardous Events

In the following we use the notation as of the table 2 below when discussing the relation between Hazardous Events in a table. For the rules defined in the coming sections we compare the two general Hazardous Events HE1 and HE2. HE1 has the general Situation A, the Hazard X, and gets the concluded ASIL attribute value ASIL1, when analysing the resulting effects on Exposure, Controllability and Severity. For HE2 the Situation is denoted B, the Hazard Y, and the resulting ASIL value ASIL2.

Table 2. General notation in Hazardous Event table used in formulation of rules

Hazardous Event ID	Situation	Hazard	Integrity Value
HE1	'A'	'X'	'ASIL1'
HE2	'B'	'Y'	'ASIL2'
HE3	'C'	'Z'	'ASIL3'
…	…	…	…

A. Rules for Identification of Dominance

There are a number of cases when adding a new Hazardous Event would not contribute to the list of already identified Safety Goals. Such Hazardous Events are of no interest, when the objective of the list of Hazardous Events is to identify the list of Safety Goals. A first obvious example is when the candidate HE has exactly the same Situation and the same Hazard as an already listed HE, but a lower ASIL value. The same applies if it is only a difference in Situation or a difference in Hazard, and the other two columns are identical. In all these cases we can directly conclude that the candidate HE will not add any Safety Goal compared to the ones already identified. Thus, we can formulate our first rule of dominance:

Rule DI: Dominance exists if two columns show relation 'identical' and the third one has the relation '⊂' or '<'.

Our next observation is that for a given Hazard, there will only become one resulting Safety Goal (this is how Safety Goals are identified). Thus if two Hazardous Events have the same Hazard, but different ASIL values, the one with the lowest ASIL value does not add anything to the analysis. This leads us to the second rule of dominance:

Rule DII: Dominance exists if Hazard show relation ’identical’ and the Integrity values are different (regardless of relation for Situation).

In a next step, we can conclude three more observations possible to aggregate into one rule. 1) A more general Situation with a higher ASIL value will dominate as long as the Hazard relation is identity. 2) A more general Hazard with a higher ASIL will dominate as long as the Situation relation not implies a subset relation in the opposite direction. 3) When both Situation and Hazard are more general, this will cause dominance if the ASIL value is not lower. We aggregate these observations into our third rule of dominance:

Rule DIII: If two or more columns have the relation ’⊂’ or ‘<‘, there is dominance if they are all in the same direction, given that the column Hazard does not have the relation ‘mutual exclusive’ or ‘overlapping’.

Finally, we have the case when the one Hazard is a special case of the other (a subset), and the more confined Hazard also has a lower ASIL value. In this case, the more general Hazard with the higher ASIL value will generate the dimensioning Safety Goal, and the other Hazardous Event will not add anything to the analysis. This will always hold, independent of how general the Situations are defined. This leads to the fourth and last rule of dominance:

Rule DIV: Dominance exists if Hazard has ’⊂’ relation in the same direction as the Integrity value has the ‘<‘ relation, regardless of the Situation relation.

For now we say that these are the only four rules of dominance needed to categorize all possible cases where one Hazardous Event can be seen as dominated by another. In a section VI, there is a proof that no other rules are needed, i.e. these four rules of dominance are complete. Before this, the rules for non-dominance are identified.

B. *Rules for Identification of Non-Dominance*

In the previous section we identified the rules identifying when one Hazardous Event makes another one unnecessary, i.e. when the other does not imply a unique Safety Goal. In order to determine the relation between any two Hazardous Events, it is as important to conclude when they both contribute to unique Safety Goals.
Our first observation regarding such so called non-dominance, is when we are comparing two mutual exclusive Hazards or two overlapping Hazards. As long as both Hazardous Events are based on a Hazard which is (partly) unique, this will also imply that it will contribute to a unique Safety Goal. We can hence formulate our first rule of non-dominance:

Rule NI: There is never dominance between two mutual exclusive Hazards or between two overlapping Hazards.

Next observation is that a more specific Hazard having a higher ASIL attribute than the general Hazard will add a new Safety Goal, but the first Safety Goal will still stay unique. For example, we can have one Safety Goal stating that we shall avoid steering angle failures above 20% by ASILD, and another one stating that we shall avoid steering angle failures above 5% by ASILC. The first one is more restrictive regarding the ASIL value, while the second is more restrictive regarding the threshold above which a deviation is considered as a failure. This means that no one of these two Safety Goals includes the other one, and they are hence both to be considered as unique. This leads us to the formulation of our second rule of non-dominance:

Rule NII: If the Hazard and Integrity relations are in different directions, there is never dominance.

Furthermore, we can observe that even if the Integrity value is identical for a pair of Hazardous Events, they will still both contribute to unique Safety Goals if either of Situation or Hazard cannot be seen as a subset of the other one. If either of the Situation or the Hazard relations, are mutual exclusive or overlapping, this implies that we cannot say that the Safety Goal derived from the one HE will include the Safety Goal derived by the other. This leads us to the formulation of the third rule of non-dominance:

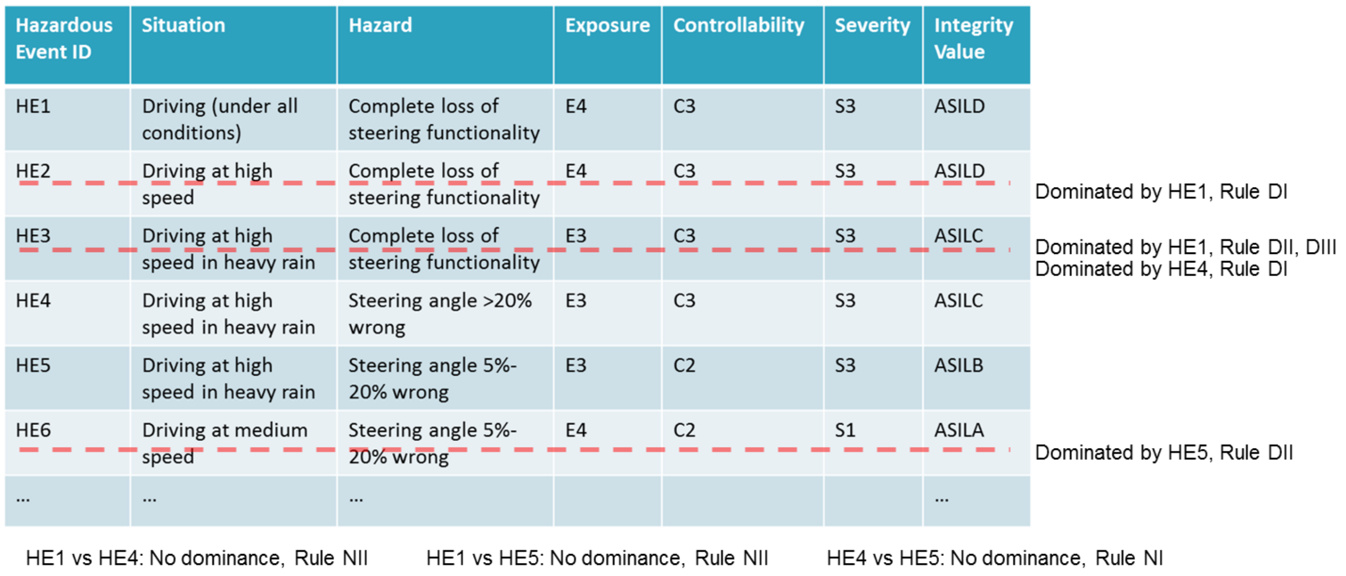
Rule NIII: There is never dominance if Integrity value relation is 'identical', and any of the other two relations are either 'mutual exclusive' or 'overlapping’.

Finally, we conclude that even if the Integrity relation between two Hazardous Events is ‘identical’, this will still imply two unique Safety Goals if the relations for Situation and Hazard both have a subset relation, but in different directions. This leads us to the conclusion of the fourth rule of non-dominance:

Rule NIV: There is no dominance if Integrity value relation is 'identical', and the other two relations are in different directions.

We have now formulated 4+4 rules to determine whether any pair of Hazardous Events will generate one or two Safety Goals. In the former case we call that dominance, and in the latter case non-dominance. In the next section we investigate to what extent these eight rules are complete and consistent. We want that all possible pairs of Hazardous Events are handled by at least one rule, and that there will never be any conflicting rules for any pair of Hazardous Events.

Table 3. Example extract of a Hazardous Event table, revisited



Hazardous Event ID	Situation	Hazard	Exposure	Controllability	Severity	Integrity Value	
HE1	Driving (under all conditions)	Complete loss of steering functionality	E4	C3	S3	ASILD	
~~HE2~~	~~Driving at high speed~~	~~Complete loss of steering functionality~~	~~E4~~	~~C3~~	~~S3~~	~~ASILD~~	Dominated by HE1, Rule DI
~~HE3~~	~~Driving at high speed in heavy rain~~	~~Complete loss of steering functionality~~	~~E3~~	~~C3~~	~~S3~~	~~ASILC~~	Dominated by HE1, Rule DII, DIII Dominated by HE4, Rule DI
HE4	Driving at high speed in heavy rain	Steering angle >20% wrong	E3	C3	S3	ASILC	
HE5	Driving at high speed in heavy rain	Steering angle 5%-20% wrong	E3	C2	S3	ASILB	
~~HE6~~	~~Driving at medium speed~~	~~Steering angle 5%-20% wrong~~	~~E4~~	~~C2~~	~~S1~~	~~ASILA~~	Dominated by HE5, Rule DII
…	…	…				…	

HE1 vs HE4: No dominance, Rule NII HE1 vs HE5: No dominance, Rule NII HE4 vs HE5: No dominance, Rule NI

VI. Completeness and Consistency of the Rules

In the previous chapter we categorize any pair of Hazardous Events as the combined effect of the relations for Situation, Hazard and ASIL value, respectively. For the Situation and the Hazard relations, there are five different possibilities each: Identical, Mutual exclusive, overlapping, and a subset relation in any of the two directions. For the ASIL value relation, there are three possibilities: Identical, and the one is higher or the other is higher. In total this implies 5*5*3=75 different possibilities to categorize the relation between any pair of Hazardous Events. Below in table 4, there is an extensive list of all these 75 possibilities, and for each is also noted what rules for dominance and for non-dominance that apply.
The first row in this table is about when the two Hazardous Events are identical, and thus no comparison is motivated. For the remaining 74 rows there is some difference between the two compared Hazardous Events. We observe that for every row in this list, there is at least one rule that is found applicable. Furthermore we observe that there is no row where there is one rule for dominance and another for non-dominance at the same time. The fact that all rows are covered by at least one rule, and that there is no row showing any contradicting rules, implies that our set of 4+4 rules is complete and consistent.

VII. Discussion

Our eight rules for determining dominance or non-dominance can be used in at least two ways. The first use case is to review a list of candidate Hazardous Events, and remove all of these that can be shown as dominated by any of the others. For the remaining Hazardous Events, it is then possible to show that they pairwise show non-dominance. The second use case is to review a list of Hazardous Events with respect to its completeness. This means that rules for non-dominance are used to identify candidates missing in the list. Let us go back to our example with LKA steering list of Hazardous Events from Table 1. As shown in the Table 3 we can now conclude that neither of the candidates HE2, HE3, and HE6 will conclude to the identification of a unique Safety Goal. Furthermore, we can also make sure that the remaining Hazardous Events, all contribute to a unique Safety Goal.
We can then continue our review by challenging this list by trying to add more Hazardous Events. However, we only add those candidates that are shown not to become dominated by one of the existing ones. We might come up with a new candidate that dominates one of the HE already in list, which implies that the new one will replace the dominated one.
In our example we can now consider all combinations of Situations and Hazards to find out whether any of these would generate a dimensioning Hazardous Event. Given that we have chosen these Situations and Hazards, respectively, and that they catch the cases when any of the E, C or S factors can change its value, we can argue for the completeness of the concluded list of Hazardous Events. Our rules give a hint which potential Hazardous Events to consider, which means that we can find arguments for a number of Hazardous Events at the time, why not to consider any of them (as they would be dominated by an existing Hazardous Event).
For example, given that we have HE1 in our list above, we can directly conclude that we do not need to look for any other Situation to combine with this Hazard, as they would all be dominated. The argument for this conclusion is that HE1 will not have a lower ASIL than any other HE, and also that no other Situation could be seen as a superset to the Situation of HE1. We can formulate it by saying that comparing HE1: <A,X,ASIL1> with any other HEk: <B,X,ASIL2> (the same Hazard), HEk will always be dominated by HE1. The argument for this is that ASIL2 is not greater than ASIL1 (ASIL1=ASILD), and B is always a subset of A (A is the most general situation). This means that either ASIL2=ASILD and then rule DI is applicable, or ASIL2 has a lower value and then rule DII is applicable. In a similar way as in the example above, a number of candidate Hazardous Events can in many situations be evaluated simultaneously.

Table 4. Investigation of all possible pairs of HE

Situation	Hazard	Integrity	Dominance	Rule(s)
Identical	Identical	Identical	–	Identity
Identical	Identical	ASIL1	HE2	Rule DI, DII
Identical	Identical	ASIL2	HE1	Rule DI, DII
Identical	Mutual	Identical	No Dominance	Rule N1, NII
Identical	Mutual	ASIL1	No dominance	Rule NI
Identical	Mutual	ASIL2	No dominance	Rule NI
Identical	X subset of Y	Identical	HE2 dominates	Rule DI
Identical	X subset of Y	ASIL1	HE2 dominates	Rule DIII,
Identical	X subset of Y	ASIL2	No dominance	Rule NII
Identical	Y subset of X	Identical	HE1 dominates	Rule DI
Identical	Y subset of X	ASIL1	No dominance	Rule NII
Identical	Y subset of X	ASIL2	HE1 dominates	Rule DIII,
Identical	Overlapping	Identical	No dominance	Rule NI, NIII
Identical	Overlapping	ASIL1	No dominance	Rule NI
Identical	Overlapping	ASIL2	No dominance	Rule NI
Mutual	Identical	Identical	No dominance	Rule NIII
Mutual	Identical	ASIL1	HE2 dominates	Rule DII
Mutual	Identical	ASIL2	HE1 dominates	Rule DII
Mutual	Mutual	Identical	No dominance	Rule NI, NIII
Mutual	Mutual	ASIL1	No dominance	Rule NI
Mutual	Mutual	ASIL2	No dominance	Rule NI
Mutual	X subset of Y	Identical	No dominance	Rule NIII
Mutual	X subset of Y	ASIL1	HE2 dominates	Rule DIII,
Mutual	X subset of Y	ASIL2	No dominance	Rule NII
Mutual	Y subset of X	Identical	No dominance	Rule NIII
Mutual	Y subset of X	ASIL1	No dominance	Rule NII
Mutual	Y subset of X	ASIL2	HE1 dominates	Rule DIII,
Mutual	Overlapping	Identical	No dominance	Rule NI, NIII
Mutual	Overlapping	ASIL1	No dominance	Rule NI
Mutual	Overlapping	ASIL2	No dominance	Rule NI
A subset of B	Identical	Identical	HE2 dominates	Rule DI
A subset of B	Identical	ASIL1	HE2 dominates	Rule DII
A subset of B	Identical	ASIL2	HE1 dominates	Rule DII
A subset of B	Mutual	Identical	No dominance	Rule NI, NIII
A subset of B	Mutual	ASIL1	No dominance	Rule NI
A subset of B	Mutual	ASIL2	No dominance	Rule NI
A subset of B	X subset of Y	Identical	HE2 dominates	Rule DIII
A subset of B	X subset of Y	ASIL1	HE2 dominates	Rule DIII,
A subset of B	X subset of Y	ASIL2	No dominance	Rule NII
A subset of B	Y subset of X	Identical	No dominance	Rule NIV
A subset of B	Y subset of X	ASIL1	No dominance	Rule NII
A subset of B	Y subset of X	ASIL2	HE1 dominates	Rule DIV
A subset of B	Overlapping	Identical	No dominance	Rule NI, NIII
A subset of B	Overlapping	ASIL1	No dominance	Rule NI
A subset of B	Overlapping	ASIL2	No dominance	Rule NI
B subset of A	Identical	Identical	HE1 dominates	Rule DI
B subset of A	Identical	ASIL1	HE2 dominates	Rule DII
B subset of A	Identical	ASIL2	HE1 dominates	Rule DII
B subset of A	Mutual	Identical	No dominance	Rule NI, NIII
B subset of A	Mutual	ASIL1	No dominance	Rule NI
B subset of A	Mutual	ASIL2	No dominance	Rule NI
B subset of A	X subset of Y	Identical	No dominance	Rule NIV
B subset of A	X subset of Y	ASIL1	HE2 dominates	Rule DIV
B subset of A	X subset of Y	ASIL2	No dominance	Rule NII
B subset of A	Y subset of X	Identical	HE1 dominates	Rule DIII
B subset of A	Y subset of X	ASIL1	No dominance	Rule NII
B subset of A	Y subset of X	ASIL2	HE1 dominates	Rule DIII,
B subset of A	Overlapping	Identical	No dominance	Rule NI, NIII
B subset of A	Overlapping	ASIL1	No dominance	Rule NI
B subset of A	Overlapping	ASIL2	No dominance	Rule NI
Overlapping	Identical	Identical	No dominance	Rule NIII
Overlapping	Identical	ASIL1	HE2 dominates	Rule DII
Overlapping	Identical	ASIL2	HE1 dominates	Rule DII
Overlapping	Mutual	Identical	No dominance	Rule NI, NIII
Overlapping	Mutual	ASIL1	No dominance	Rule NI
Overlapping	Mutual	ASIL2	No dominance	Rule NI
Overlapping	X subset of Y	Identical	No dominance	Rule NIII
Overlapping	X subset of Y	ASIL1	HE2 dominates	Rule DIII,
Overlapping	X subset of Y	ASIL2	No dominance	Rule NII
Overlapping	Y subset of X	Identical	No dominance	Rule NIII
Overlapping	Y subset of X	ASIL1	No dominance	Rule NII
Overlapping	Y subset of X	ASIL2	HE1 dominates	Rule DIII,
Overlapping	Overlapping	Identical	No dominance	Rule NI, NIII
Overlapping	Overlapping	ASIL1	No dominance	Rule NI
Overlapping	Overlapping	ASIL2	No dominance	Rule NI

VIII. Conclusion

We have defined eight rules to be used for the identification of a minimal set of Hazardous Events necessary to identify all Safety Goals of an Item. The rules are used to compare any two candidates of Hazardous Events to conclude whether they are both generating a unique Safety Goal, or whether the one Hazardous Event can be seen as uninteresting (dominated by the other).

The rules are based on a categorization of the Situations, the Hazards and the ASIL attribute values, respectively. Regarding the ASIL attribute values, the integrity levels are either equal, or one of them is higher than the other. For both Situations and for Hazards, we use set theory to describe any pairwise relation. We show that our eight rules are complete and consistent. The completeness is shown as any possible combination of relations between Situations, Hazards, and ASIL attribute value, is covered by at least one rule. Consistency is shown as none of these possible combinations implies both dominance and non-dominance. This means that any combination is uniquely identified as either dominance or non-dominance.

This set of rules makes it possible to solve the paradox of being specific in the list of Situations and Hazards, and still end up with a limited number of dimensioning Hazardous Events. Today, many companies fear to be too detailed in the Hazard Analysis, as it might generate a potentially infinite number of Hazardous Events. Instead they run the risk of becoming unnecessarily conservative in the analysis, leading to a too expensive product. By applying a methodology where these eight rules are applied in the generation and the review of Hazardous Events, it is feasible to generate a list that is at the same time complete and precise.

References

[1] ISO, 26262-3:2011, Road vehicles — Functional safety — Part 3, Concept phase, 2011.

[2] Birch, J., Rivett, R., Habli, I., Bradshaw, B., Botham, J., Higham, D., Jesty, P., Monkhouse, H., Palin, R., "Safety Cases and their role in ISO 26262 Functional safety Assessment", SAFECOMP, Toulouse, 2013.

[3] Jesty, P.H., Ward, D.D., Rivett, R.S., ""Hazard Analysis for programmable automotive Systems", Proceddings of 2nd International Conference on system Safety, IET, 2007.

[4] Blanquart, J-P, Astruc, J-M, Baufreton, P., Boulanger, J-L., Delseny, H., Gassino, j., Ladier, G., Ledinot, E., Leeman, M., Machrouh, J., Quere, P., Ricque, B., "Criticality categories across safety standards in different domains", ERTS2 congress on embedded real time system and software, Toulouse, 2012.

[5] Jang, H.A., Hong, S-H, Lee, M.K., "A Study on situation analysis for ASIL Determination", Journal of Industrial and Intelligent Information Vol. 3, No. 2, 2015.

[6] Verband der Automobilindustrie e.V. (VDA), "Situationskatalog E-Parameter nach ISO 26262-3", 2015.

[7] Martin, H., Winkler, B., Leitner, A., Thaler, A., Cifrain, M., Watzenig, D., "Investigation of the influence of non-E/E safety measures for the ASIL determination", 39th Euromicro Conference Series on software Engineering and advanced Applications, 2013.

[8] Johansson, R, "The Importance of Active Choices in Hazard Analysis and Risk Assessment", CARS 2015 - Critical Automotive applications: Robustness & Safety, 2015.

Session 21

Model Driven Engineering In practice 1

Thursday 28th, 16:30 – Auditorium St Exupery

Architecture-led Diagnosis and Verification of a Stepper Motor Controller

Peter H. Feiler, Charles B. Weinstock, John B. Goodenough, Julien Delange, Ari Z. Klein, Neil Ernst
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA
[phf|weinstock|jbg|jdelange|azk|nernst}@sei.cmu.edu
Keywords: Architecture modeling & analysis, fault analysis, embedded software system

ABSTRACT

This paper discussed an architecture-led approach to diagnosing time sensitive issues with a stepper motor controller that manages fuel flow of an engine. A real engine control system design had originally been modeled and verified with SCADE®. The potential for missed steps that result in misalignment in the fuel valve position is difficult to test for and was not discovered until after the engine went into operation. We utilize the execution and communication timing semantics of AADL to architecturally characterize the interaction between the elements of the stepper motor control systems. We then characterize the functional behavior in the context of the task dispatch and input handling semantics using the AADL Behavior Annex and identify potential fault sources and their impact using the AADL Error Model Annex. The identified the potential error sources, early arrival and mismatched command rates, we quantify the condition for this to occur and analyze the system based on timing data from scheduling analysis and actual timing measurements. We use this analysis to evaluate several proposed design corrections.

I. INTRODUCTION

This case study shows how an analytical architecture fault-modeling approach can be combined with static analysis techniques to diagnose a time-sensitive design error in a control system and to verify that proposed changes to the system address the problem. The analytical approach demonstrates the value of the SAE Architecture Analysis & Design Language (AADL) standard [1] with its well-defined timing and fault behavior semantics in discovering hard-to-test errors and correcting them early in the life cycle, thereby reducing rework cost. This virtual system integration approach is a key element of an architecture-centric framework for improving the qualification assurance of software-reliant safety-critical systems [2].

In this case study, we investigated an actual stepper-motor system (SMS) that is part of an aircraft engine control system that manages fuel flow by adjusting a fuel valve. The original design was developed and verified in a model-based development environment called SCADE Suite®, and an implementation was tested on actual equipment. In some test situations, actual fuel flow did not correspond to the desired fuel flow. The failure was suspected to be due to execution time jitter in the stepper-motor control system, which resulted in some steps being missed. Missed steps were not immediately detectable by the controller to take corrective action. Two repairs were proposed to correct the problem, but there was little evidence other than testing that either proposed solution would address the problem of missed steps. We use the same analysis techniques to diagnose the problem and to determine whether the proposed design changes addressed the problem. A full case study report elaborates on how to manage the results of the diagnostic analysis and verification as an assurance case [3].

We first describe the SMS and characterize its architecture as an AADL model. We then diagnose the SMS in three steps: discuss the behavioral verification of the different SMS elements to eliminate computational errors; discuss the fault analysis of the SMS with focus on time sensitive issues, but also supporting a full fault impact analysis; and discuss a quantification of the timing condition under which the potential for missed steps can occur. We then proceed with applying this analysis to proposed design changes to correct the problem.

II. THE STEPPER MOTOR SYSTEM (SMS)

The stepper-motor control system operates open loop (i.e., there is no direct feedback on the successful execution of a step by the motor). The enclosing engine control system can detect deviation from desired fuel flow, but not at the granularity of individual stepper motor steps. In other words, the problem is not detected until multiple steps are missed.

The SMS is commanded to open the fuel valve in terms of a percentage with zero being closed and 100 being completely open. The stepper motor takes a known number of steps to move the fuel valve from a completely closed to a completely open position. The SMS is expected to reach the commanded position within a bounded time that is proportional to the distance between the current position and the desired position. At command completion the stepper motor is expected to have reached the commanded position closest to the requested opening percentage.

The position control system (SM_PCS) operates periodically, and converts the percentage requested into the desired position in terms of stepper motor steps. It then commands the actuator to move the stepper motor a specified number of steps in the open or close direction. The maximum number of steps that can be taken per frame is a function of the frame rate and the time it takes the motor to move one step. To move the fuel valve as quickly as possible to the new position—that is, in a time roughly proportional to the number of steps required to move from the current position to the desired position—the position-change command sequence passed to the actuator consists of a sequence of maximum step count commands followed by a single command with the remaining steps less or equal to the maximum step count.

SM_PCS maintains a record of the desired position and the position to be reached through the most recent position-change command (commanded position). On completion of the position command, the desired position, commanded position, and actual position of the motor are expected to be the same. A *homing* command (to a fully closed position) is executed during initialization to synchronize the actual position with the initial desired and commanded position assumed by the SM_PCS.

The SMS may receive a new command before it completes the previous command. SMS is expected to immediately respond to the new command (i.e., immediately moves the fuel valve to the most recent commanded position without first continuing to the previously commanded position). As we will show this immediate response feature is not the culprit. Instead, the behavior of the actuator is sensitive to command arrival timing.

The SMS was implemented according to the above design and it was discovered that the expected location of the stepper motor deviated from the actual location of the stepper motor over time (that is, steps were missed). The functionality of the stepper motor had been modeled and verified using SCADE®, but without detecting the potential missed steps problem.

III. SMS MODELING IN AADL

The SMS was modelled in AADL in three levels of abstraction:

- SMS as the system of interest in its operational context to capture the commanded input and expected result of the controlled system,
- the runtime architecture of the SMS as a set of interacting tasks to capture the execution and communication timing semantics of the implementation to analyze the time-sensitive nature of the problem (Figure 1),
- a specification of SMS component states and functional behavior that provides the basis for quantifying the conditions under which some commanded steps may be missed.

The SMS and the Engine Control System (ECS) are represented as AADL system components. This allows us to decompose the SMS as necessary to elaborate its architecture. The ECU is represented as an AADL processor, and the device bus for transferring data between any sensor, the actuator, and the processor as an AADL bus. The power supply is also modeled as an AADL bus, in this case transferring electricity. The fuel valve is represented as an AADL device.

The SMS consists of three components: digital position control software for the stepper motor SM_PCS, an actuator SM_ACT that translates commands from the position control software into electrical signals to a stepper motor, and the stepper motor SM_Motor. Note that the commands received by SM_ACT (*Commanded_Position*) take the form of a step count to be completed within a frame in the original system design. The interface between SM_ACT and SM_Motor (*SM_Command_Signals*) is represented by a feature group, i.e., a collection of event ports. The mechanical interface of the stepper motor (*Mechanical_Control_Position*) is represented by an abstract feature, i.e., a feature without specific communication timing semantics that are associated with ports.

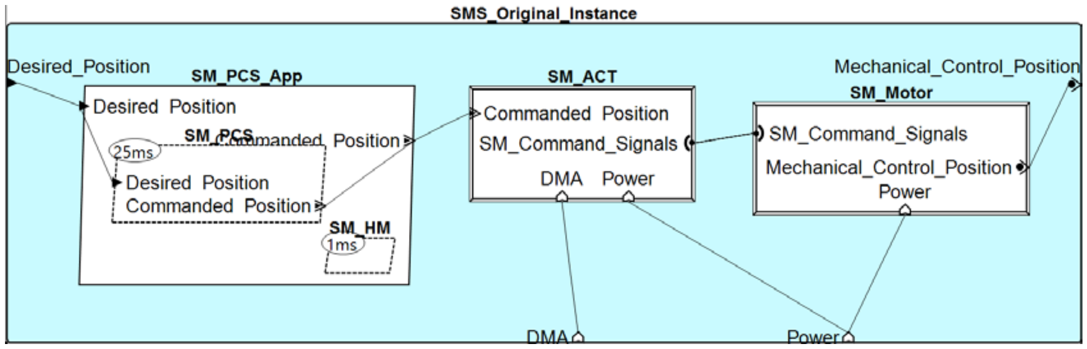


Figure 1: The SM_PCS Architecture

The position control system software SM_PCS is an AADL thread with a period of 25ms that resides in an AADL process called SM_PCS_App. The figure also shows a health monitor (SM_HM) thread with a period of 1ms in the same process that has no logical interaction with SM_PCS. This indicates that the two threads share the same address space; thus, a coding error in one can potentially affect the other.

A Binding property in the operational environment of SMS, which contains the ECU, indicates that SM_PCS and SM_HM execute on the ECU. A Priority property indicates that SM_HM takes precedence over SM_PCS, thus, can affect the completion time of SM_PCS due to preemption. A Scheduling_Protocol property on the ECU indicates that preemptive scheduling is used.

The SM_ACT and SM_Motor are modeled as AADL devices to reflect that they are separate physical components. We can specify the execution and communication timing behavior of devices the same way as for threads. We specify that the actuator responds immediately to a new command from SM_PCS. We reflect this in the AADL specification of SM_ACT (Figure 2) by the *aperiodic* dispatch protocol. The *Queue_Size* and *Overflow_Handling_Protocol* properties allow us to specify details of handling arrival of input (see Section IV.B). Similarly, the stepper motor immediately responds to each step signal from the actuator – expressed by *aperiodic* dispatch.

```
device SM_ACT
features
-- logical interface
  Commanded_Position: in event data port StepCount { Queue_Size => 0;
    Overflow_Handling_Protocol => Error; };
  SM_Command_Signals: feature group inverse of SM_Command_Signals;
-- physical interface
  DMA: requires bus access DirectAccessMemory;
  Power: requires bus access Power_Supply.Volt28;
flows
  flowpath : flow path Commanded_Position -> SM_Command_Signals;
properties
  Dispatch_Protocol => Aperiodic;
end SM_ACT;
```

Figure 2: The SM_ACT Interface Specification

IV. Diagnosis of the SMS

We diagnose the SMS problem in three steps. First, we establish a record of the functional behavior verification and its assumptions. Second, we utilize the fault modeling capability including a taxonomy to identify potential fault contributors and refine the AADL model to more fully capture the timing behavior of the SMS. Third, we formalize the time-related condition under which the problem can occur.

A. *Behavioral Verification of the SMS*

The original behavior specification was modeled and verified in the SCADE Suite®. This verification asserts that the desired, commanded, and actual positions of SM are correctly maintained. However, verification tools such as the model checker in SCADE® or Simulink® make assumptions about synchronous execution behavior and timing. Typically, execution timing is separately verified through scheduling analysis and benchmark measurements of code. For example, TAXYS [4] combines verification and code generation from Esterel models (predecessor to SCADE) with timing verification based on timed automata. Recent research in verification of time-sensitive applications includes model checking of specifications extracted from source code [5] and verification of time sensitive behavioral constraints of AADL models annotated with BLESS [6]. We have created a BLESS specification of SMS, but were faced with the challenge to represent with multiple execution rates and aperiodic execution behavior.

To better understand the effects of execution and communication timing of the different elements of SMS affects functional behavior we elaborate the AADL specification with AADL Behavior Annex (BA) annotations [7]. These annotations specify how the functional behavior originally specified in SCADE interacts with the execution and communication timing behavior of the SMS components. This mapping into the AADL tasking model allows us to diagnose where the synchronous execution model of SCADE is potentially violated.

The behavior specification of SM_PCS is shown in Figure 3 and is equivalent to the SCADE model. It maintains two states: *DesiredPositionState* to represent the target position received by the last command, and *CommandedPositionState* to represent the position resulting from the execution of the last command sent to SM_ACT. It states that on every periodic dispatch SM_PCS will

1) check whether a new *Desired_Position* command has arrived, validate that the value is within the expected range of 0 to 100, and set it to be the *DesiredPositionState*; an out of range command is ignored.

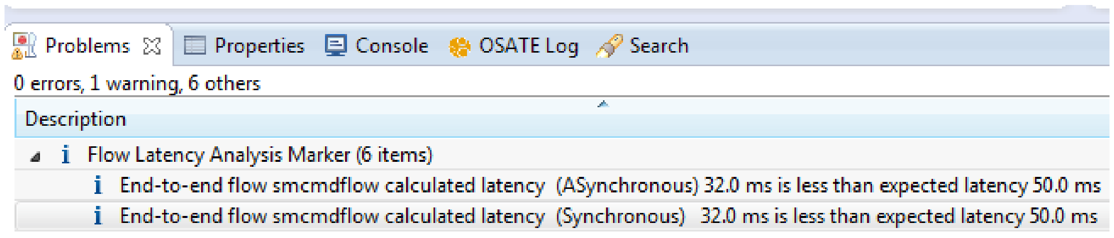
2) compute a step count up to a maximum of 15 steps per frame (SPF) in the appropriate direction to move the *CommandedPositionState* towards the *DesiredPositionState*; and issue the appropriate step count command to SM_ACT; this results in a series of 15 step command, with the last non-zero count possibly less than 15, and then a count of zero once the desired position is reached.

```
thread implementation SM_PCS.impl
subcomponents
  DesiredPositionState: data SM_Position;
  CommandedPositionState: data SM_Position;
annex Behavior_Specification {**
  variables
    distance: Base_Types::Integer;
    stepcount: Base_Types::Integer;
  states
   Ready: initial complete state;
  transitions
    Ready -[on dispatch]-> Ready { -- on every 25ms dispatch
    -- check for new command and out of range if a new command has been received
    if ((Desired_Position'fresh = true) and (Desired_Position >= 0 )
        and ( Desired_Position <= PCSProperties::MaxPercent)){
    -- convert from PercentOpen to Steps
      DesirePositionState := PCSProperties::MaxPosition*Desired_Position/100
    } end if;
    distance := DesiredPositionState - CommandedPositionState ;
    if (abs(distance)> PCSProperties::MaxStepCount)
      stepcount := PCSProperties::MaxStepCount
    else
      stepcount := abs(distance)
    end if;
    if (distance>0){
      Commanded_Position := stepcount;
      CommandedPositionState := CommandedPositionState + stepcount
    } else {
    -- this case handles steps in the close direction as well as zero steps
    -- note that zero step commands are expected to be issued
      Commanded_Position = - stepcount;
      CommandedPositionState = CommandedPositionState - stepcount
    } end if;
    Commanded_Position!;
  }; -- end action
**};
end SM_PCS.impl;
```

Figure 3: Behavior Specification of SM_PCS

Both the SCADE model and the AADL BA specification of SM_PCS have been verified to correctly send a command sequence to SM_ACT. The verification also showed that the arrival of a new desired position command before the previous desired position has been reached is handled correctly, i.e., that SM_PCS responds within one frame to the new command and sends a command sequence to actually reach the latter desired position. The verification results show that the SCADE and BA specifications reflect the same behavior with respect to the verified invariant.

Furthermore, we have used an end to end flow specification in the AADL model and performed latency analysis [8] to determine that the system responds "immediately" to the new command, the amount of time it takes for SMS to respond to the new command, i.e., that it responds within less than two frames as shown in Figure 4. This latency includes the one frame sampling delay by SM_PCS and the processing and communication time by SM_PCS and SM_ACT to issue the first step command to the motor.



```
0 errors, 1 warning, 6 others
Description
  i  Flow Latency Analysis Marker (6 items)
     i  End-to-end flow smcmdflow calculated latency  (ASynchronous) 32.0 ms is less than expected latency 50.0 ms
     i  End-to-end flow smcmdflow calculated latency  (Synchronous)  32.0 ms is less than expected latency 50.0 ms
```

Figure 4: End-to-End Latency Analysis Results for New SMS Command

The specification of the actuator as a physical device is a little more interesting in terms of timing. In this case, we reflect in the BA specification that SM_ACT responds to two inputs, the arrival of the next step count from SM_PCS and the completion of a step from the motor. This is determined by the specification of SM_ACT as an aperiodic thread with a queue size of zero for arriving commands. The SCADE model had assumed that the commands arrive at the frame rate of 25ms and that at that time the previous command had been completed.

The actuator maintains system state in the form of a persistent *StepsToDo* count as shown in Figure 5. This count is set to the step count value as soon as a command is received from SM_PCS. The actuator then sends individual step signals to the motor at the specified rate until the count is zero. For that purpose, the specification is characterized by three states:

1. *Ready*, indicating that it is waiting for a command from SM_PCS
2. *WaitOnStep* to indicate that the execution of a step by SM is in progress
3. *Decide* as an intermediate state dealing with the decision of whether there are steps left to be executed by SM.

Arrival of a *Commanded_Position* in the form of a step count is handled by the *Ready* state and the *WaitOnStep* state. *StepsToDo* is set to the newly arrived value. A positive value results in increasing the stepper motor position, while a negative value results in a decrease. The first transition out of the *Decide* state determines *that* no step has to be taken. The other transition out of the Decide state specifies whether an *Increment_Step* or *Decrement_Step* signal is to be issued according to the specified Direction and updates the step count. The transition out of the *WaitOnStep* state triggered by the step completion signal leads to the *Decide* state, which determines whether additional steps are to be performed.

```
device implementation SM_ACT.impl
subcomponents
  StepsToDo: data StepCount;
annex Behavior_Specification {**
  states
    Ready: initial state;
    WaitOnStep: complete state;
    Decide: state;
  transitions
    Ready -[on dispatch Commanded_Position]-> Decide {
      StepsToDo := Commanded_Position
    };
    WaitOnStep -[on dispatch Commanded_Position]-> WaitOnStep {
      StepsToDo := Commanded_Position
    };
    WaitOnStep -[on dispatch SM_Command_Signals.StepDone]-> Decide ;
    Decide -[StepsToDo = 0]-> Ready ;
    Decide -[StepsToDo > 0]-> WaitOnStep {
      If StepsToDo > 0
        StepsToDo :=  StepsToDo - 1;
        SM_Command_Signals.DoIncrement!;
      else if StepsToDo < 0
        StepsToDo :=  StepsToDo + 1;
        SM_Command_Signals.DoDecrement!;
      end if
    };
**};
end SM_ACT.impl;
```

Figure 5: The Functional Behavior of the Actuator

As mentioned above the SCADE model assumed that the previous command had completed, i.e., the *StepToDo* count is zero at the time the new command arrives. The verification of the BA specification shows that in the *Ready* state the *StepsToDo* count is always zero. In the case of *WaitOnStep* the count may be non-zero unless the last step has been issued. The verification shows that if the precondition of a zero *StepsToDo* count does not hold the number of individual step commands to the motor will deviate from the incoming step count command sequence.

B. Fault Analysis of the SMS

We use a fault taxonomy that is part of AADL Error Model Annex specification [9,10] to systematically identify potential contributors to missed steps. We do so by annotating every incoming and outgoing port with error propagation types as shown in Figure 6. The Error Model Annex comes with fault taxonomy to identify omission, commission, value, timing, rate, sequence, replication, concurrency, authentication, and authorization error types. The notation lets us use guidewords relevant to the domain as aliases to the more abstract terms used in the taxonomy, e.g., missing command as alternative to omission.

Since the functional behavior of SM_PCS has been verified we specify that certain errors are not expected to be propagated. Figure 6 shows the actuator assuming that the step count is within range and that the command does not reflect an incorrect position. For SM_PCS we also specify that incoming commands with out of range values are mapped into *missed commands* to reflect the behavior specified in Figure 3.

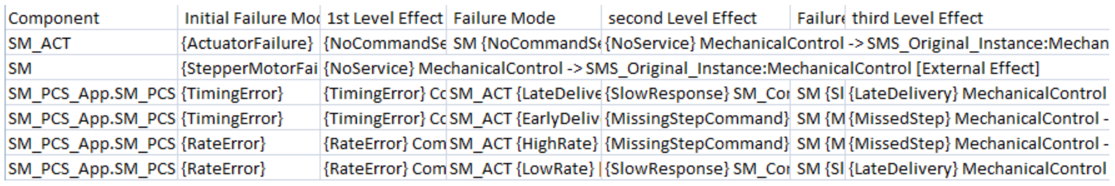
To diagnose the problem, we focus on timing related error propagations. We specify that early or late command delivery or a command sequence at an incorrect rate may occur (see Figure 6). Late command delivery delays the commanding of SM_ACT and indirectly the stepper motor, i.e., we specify an incoming late delivery is propagated as outgoing slow response by the stepper motor. Similarly, arrival of commands at a rate lower than expected results in slower response by the stepper motor.

```
annex EMV2 {**
use types SMErrorTypes;
error propagations
  Commanded_Position : in propagation { MissingStepCountCommand, TimingError, RateError};
  Commanded_Position : not in propagation { StepCountOutOfRange, IncorrectPosition};
  SM_Command_Signals : out propagation {MissingStepCommand, SlowResponse, NoCommandSequence};
  ElectricalPower : in propagation {PowerLoss};
flows
  MissingCmd: error path CommandedPosition{MissingCommand} -> SM_Command_Signals{MissedSteps};
  LateCmd: error path CommandedPosition{LateDelivery} -> SM_Command_Signals{SlowResponse};
  EarlyCmd: error path CommandedPosition{EarlyDelivery} -> SM_Command_Signals{MissingStepCommand};
  Fast: error path CommandedPosition{ HighRate} -> SM_Command_Signals(MissingStepCommand);
  Slow: error path CommandedPosition{ LowRate} ->  SM_Command_Signals(SlowResponse);
  MechanicalFailure: error source SM_Command_Signals{NoCommandSequence} when {ActuatorFailure};
  NoPower: error path Power{PowerLoss} -> SM_Command_Signals{NoCommandSequence};
end propagations;
**};
end SM_ACT;
```

Figure 6: Fault Propagation Specification for the Actuator

Early command arrival at SM_ACT is the interesting case. The specification of SM_ACT in Figure 2 indicates that input is not queued (*Queue_Size* of zero in Figure 2). In other words, SM_ACT responds to the arrival immediately. The BA specification in Figure 5 shows that the new step count is assigned to *StepsToDo* at arrival. As discussed in the previous section this potentially leads to overriding a non-zero count. We reflect this behavior in the SM_ACT interface specification by the *Overflow_Handling_Protocol* property value of *Error* for the incoming port of SM_ACT (see Figure 2). Based on this observation we proceed in the next section to define the condition in terms of time for which the *StepsToDo* count is non-zero.

The fault propagation specification for SM_ACT shown in Figure 6 includes physical failures as well, e.g., the mechanical failure of the actuator device, and the loss of electrical power from a power source external to the actuator (and SMS). Once we have completed such fault propagation specifications for each of the SMS component we can perform a fault impact analysis [11] and identify other potential contributors to missed steps or other stepper motor malfunction. Figure 7 shows a portion of a fault impact report for SMS.



Component	Initial Failure Mo	1st Level Effect	Failure Mode	second Level Effect	Failure	third Level Effect
SM_ACT	{ActuatorFailure}	{NoCommandSe	SM {NoCommandSe	{NoService} MechanicalControl -> SMS_Original_Instance:Mechan		
SM	{StepperMotorFai	{NoService} MechanicalControl -> SMS_Original_Instance:MechanicalControl [External Effect]				
SM_PCS_App.SM_PCS	{TimingError}	{TimingError} Cc	SM_ACT {LateDelive	{SlowResponse} SM_Cor	SM {Sl	{LateDelivery} MechanicalControl
SM_PCS_App.SM_PCS	{TimingError}	{TimingError} Cc	SM_ACT {EarlyDeliv	{MissingStepCommand}	SM {M	{MissedStep} MechanicalControl -
SM_PCS_App.SM_PCS	{RateError}	{RateError} Com	SM_ACT {HighRate}	{MissingStepCommand}	SM {M	{MissedStep} MechanicalControl -
SM_PCS_App.SM_PCS	{RateError}	{RateError} Com	SM_ACT {LowRate} |	{SlowResponse} SM_Cor	SM {Sl	{LateDelivery} MechanicalControl

Figure 7: Fault Impact Report for SMS

C. *Formalized Time Sensitve Fault Condition*

Once we understand the issue of overriding a non-zero *StepsToDo* count, we can quantify how early a command must arrive for this condition to occur. The latest time for a non-zero value is when the last step of a position-change command is issued. Since the actuator is a reactive component, the maximum early arrival time between two commands from SM_PCS must not exceed the difference of the period of 25ms and this time limit.

This leads to the following condition that must be satisfied in order to avoid a missed step. The maximum early arrival time for commands arriving at SM_ACT must be less than the time difference between the latest time for a non-zero step count value and the next frame, which we refer to as *AStepMissBound*..

```
Max(EarlyArrivalTime) < StepMissBound
```

The maximum early arrival time is determined by the maximum early send time by SM_PCS and variation in communication time. Figure 8 illustrates how the maximum early send time is determined. The time difference between a send at maximum completion time followed by a send at minimum completion time is the frame time minus the delta between maximum and minimum completion time. Thus, the early send time corresponds to this delta. This is reflected in the formula

```
Max(EarlyArrivalTime) = Delta(CompletionTimeSM_PCS) + Delta(Comm)
```

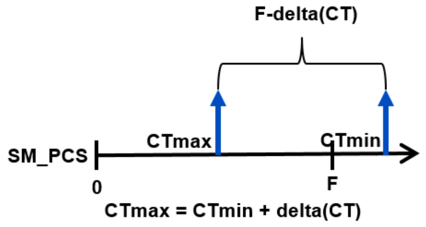


Figure 8: Maximum Early Send Time to Actuator

The value of step count is non-zero until the last step of a position-change command has been issued (i.e., until (Step_Count -1) * Step_Duration). This results in a worst-case *step miss bound* for the maximum acceptable variation of inter-arrival time of

```
StepMissBound = 25ms – ((MaxStepCount -1) * max(Step_Duration))
```

According to the stepper motor specification the step duration varies between 578 (1.730ms step duration) and a maximum of 621 steps (1.61ms step duration). Using the maximum step duration as worst case, this results in a *step miss bound* of 0.78ms.

For a particular set of execution times for SM_PCS and SM_HM we can calculate the inter-arrival time variation. The worst-case send time variation corresponds to the variation in execution completion time for SM_PCS. The completion time can be determined by a scheduling analysis and confirmed by actual timing measurements of executing code. Effectively, the completion time is determined by the variation in actual execution time of SM_PCS plus any preemption variation by SM_HM. Note that preemption variation by SM_HM may be multiples of SM_HM execution time as the minimum and maximum number of preemptions may differ. The resulting number is compared to the bound of 0.78ms to determine the possibility of missed steps.

A rate mismatch between the sender SM_PCS and the receiver SM_ACT can occur for two reasons:

- SM_PCS executes at a rate faster than the specified 25ms frame rate
- SM_ACT requires more than 25ms to complete the execution of the position-change command.

SM_PCS can execute faster than 25ms if the hardware clock of the ECU operates faster. For our case study we assume that this is not the case.

Notice that this bound is less than the minimum duration of one step execution. In other words, the last step may not complete before the end of the 25ms frame. Recall that the data sheet for the stepper motor indicates that the stepper motor's step duration varies between 578 and 621 steps per second when executing at the rate of 15 steps per frame. This results in a SM_ACT completion time variation between 24.15ms and 25.95ms for performing 15 steps. In other words, SM_PCS may send commands at a higher rate than SM_ACT is processing them.

In a worst-case scenario, the stepper motor could continuously operate at the longest step duration, falling behind 0.95ms for every frame that executes a position-change command of 15 steps. The completion delay is cumulative for a sequence of consecutive maximum step count commands. Note that SM_PCS sends the maximum step count only until the desired position is reached. A step count less than the maximum allows the stepper motor to catch up with the SM_ACT commands and make up for the time delay.

This leads to a derived requirement for the SMS implementation.

```
max(StepDuration) * MaxStepCount * max(MaxStepCountCommandSequenceLength) < StepMissBound
```

It takes 16 commands (250/15) to go from a completely closed to a completely open position with a cumulative delay of 15.2ms. This number is larger than the step miss bound, leading to missed steps.

To make matters worse, the maximum step count sequence potentially can be longer. A new desired position may be issued before the previous one has been reached. The new position may be in the opposite direction from the current position. As a result, the sequence of maximum step commands can be larger than 16 step count commands. In this case the cumulative delay may exceed a frame, resulting in missing a complete step count command.

V. Verification of Alternative Designs

Two possible design changes had been proposed to address the missed step problem. The first proposed design change was to minimize output jitter by the SM_PCS by a second periodic thread sending the output at a fixed offset from its dispatch time (*Fixed Send Time* solution). The offset was chosen to be half the period as this would allow an implementation with a single thread executing at double the rate to alternate between computing the step count and sending the resulting command. In the AADL model we chose to specify a second thread with an offset start time. When analyzing this design alternative, we determine that early arrival time has been reduced but not eliminated. However, we have additional thread dispatches. Furthermore, we have to assure that the computation of SM_PCS completes before the chosen output time. Otherwise, the old step count value may be sent again. Finally, the solution does not address rate errors.

The second proposed design change was for the actuator to buffer the incoming command until the execution of the previous command is completed (*Buffered Command* solution). This can be accomplished by actually buffering the command or by adding the incoming step count to the *StepsToDo* count. If we choose command buffering it is natural to assume a queue size to be sufficient since commands are issued at the frame rate. However, the rate error analysis has shown that there is potential for cumulative delay. The addition of the step count to the *StepsToDo* count requires more complex functional logic in the actuator device. For this solution, we can verify that the count is updated correctly, even when the direction changes. The solution is less sensitive to rate errors. Cumulative delay reduces the responsiveness of the stepper motor.

In the original design and the two proposed design changes SM_PCS sends a step count to the actuator. This step count represents a state change, i.e., the difference between the current state and a new state. The full fault analysis considers the potential of data corruption or loss when the step count is communicated to the actuator. Communication of state change is sensitive to such faults, i.e., results in missed steps or execution of an incorrect number of steps. Therefore we consider an architecture design of SMS where the state, i.e., the target position is communicated to the actuator.

In in this design the desired position is validated by SM_PCS and then passed to the actuator. The functional logic of the actuator is slightly more complex, i.e., it has to compare to values instead of testing a single value for zero. However, the complexity of SM_PCS is significantly reduced. We have eliminated the functional logic of SM_PCS to transform the desired state (Desired Position) into a sequence of state changes (step count). Note that a design that operates with state changes assumes guaranteed communication and execution of every state change. In other words, it is sensitive to malfunction such as incomplete execution or data corruption during data transfer. An architecture design that communicates the desired state more robust, e.g., transient data corruption during transfer does not lead to a permanently inconsistent state.

In the case study we have considered not only logical design defects in the SMS, but also the effect of other contributors to missed steps. Table 1 presents a comparison of the four architecture design alternatives in terms of their full fault analysis. The first row focuses on logical failures in the SMS design, the second row describes mechanical failures within the SMS, the third row captures the effects of computer hardware on the SMS, and the last row represents mechanical failures in the operational environment.

The comparison shows that the position-commanded actuator design is not sensitive to early delivery or high rate errors, nor is it sensitive to transient message corruption or loss, while the original design is sensitive to transient data corruption. This is due to the design choice of commanding the actuator by desired position rather than by a sequence of position-change commands.

We can also see that mechanical failures affect the SMS the same way in all designs and must be addressed at the enclosing system level (e.g., by replication of the engine control system and the engine).

Missed Step	**Original Design**	**Fixed Send Time**	**Buffered Command**	**Position Command**
SMS logical failures	EarlyDelivery HighRate	HighRate	HighRate	
SMS mechanical failures	ActuatorFailure StepperMotorFailure	ActuatorFailure StepperMotorFailure	ActuatorFailure StepperMotorFailure	ActuatorFailure StepperMotorFailure
Transient comm failures	MessageCorruption MessageLoss	MessageCorruption MessageLoss	MessageCorruption MessageLoss	
Mechanical failures in Op Environment	ECUFailure PowerLoss ValveFailure	ECUFailure PowerLoss ValveFailure	ECUFailure PowerLoss ValveFailure	ECUFailure PowerLoss ValveFailure

Table 1: Comparison of Architecture Design Alternatives

VI. Conclusion

The purpose of this case study was to show how architecture fault modeling and analysis can be used to diagnose a time-sensitive design error encountered in a control system and to investigate whether proposed changes to the system address the problem. The analytical approach demonstrates that such errors that are hard to test for can be discovered and corrected early in the life-cycle, thereby reducing rework cost.

The case study example is a stepper motor controller to manage the fuel flow of an engine. Its original design had been verified with SCADE® without discovering until system integration and operational testing the potential for missed steps due to variation in command inter-arrival time. The use of models to capture the behavior of a system and their verification through simulation or model checking is an established practice. For time sensitive applications these models assume a particular execution model, e.g., a periodic sampling processing model with deterministic sampling behavior. Scheduling analysis is used to assure that a set of tasks are schedulable, i.e., the tasks meet their deadline. Application code and the runtime executive may be generated and configured from such verified models to ensure consistency. The resultant system still goes through system integration and operational tests.

We have presented an architecture-led approach that is more comprehensive in utilizing model-based analysis early in development and as diagnostic tool. Our unique contribution is to complement the above mentioned techniques with a combination of AADL BA, and EMV2, to represent the system, utilize a fault taxonomy to identify potential faults, and quantify timing related faults.

AADL captures a specification of the task and communication behavior of software as well as hardware devices in AADL that captures both synchronous and asynchronous system execution behavior of software and physical devices. BA associates functional behavior specification with the task and communication model, which allows us to identify mismatched assumptions about execution and communication timing semantics. We utilize a fault taxonomy and EMV2 annotations of AADL models to identify potential issues and analyze their impact throughout the system. We quantify timing related conditions that violate a behavioral assumption about command completion and utilize scheduling analysis result of variability between best case and worst-case completion times to assess whether this condition can occur.

To diagnose the time sensitive nature of the problem we have captured the original SMS architecture design and three design alternatives in AADL, the Behavior Annex, and the EMV2 Annex, and quantified a timing related condition due to early rather than late arrival times that allows us to analytically assess whether the condition can actually occur.

The ability of AADL abstractly capture the dispatch and input handling of software threads like the stepper motor controller and physical devices like the actuator helped us focus on the essential architecture aspects of the system. The ability to specify the functional behavior of each component in the context of its dispatch and input handling behavior allowed us to recognize command execution is aborted due to the fact that a counter is set to a new target value even under circumstances when it contains a non-zero value. We have applied the fault taxonomy of the AADL Error Model Annex to perform a full safety analysis that includes logical design errors as well as physical errors. This fault taxonomy includes error types that deal with the time-sensitive nature of systems, both in terms of early or late arrival and in terms of mismatched arrival rates. We have been able to quantify the condition for timing related faults. We have then used results from scheduling analysis or actual timing measurements to analytically determine whether and when the missed step failure can occur. We have shown that in addition to early arrival, rate mismatch can lead to missed steps in the operation of the stepper motor.

We have applied the analysis to the original design as well as the three design alternatives. Three design communicate state change, i.e., the number of steps to be performed, while the fourth communicates state, i.e., the target position. During fault analysis we have identified system designs that involve state change to be more sensitive to transient faults such as data corruption or message loss. They result in persistent incorrect state for the receiver. When communicating complete state repeatedly, transient data corruption and message loss is limited to transient effects. This leads to a more robust system design.

VII. References

1. Peter H. Feiler, David P. Gluch, Model-Based Engineering with AADL - An Introduction to the SAE Architecture Analysis and Design Language. Addison-Wesley Professional, 2012.
2. Peter H. Feiler, John B. Goodenough, Arie Gurfinkel, Charles Weinstock and Lutz Wrage, "Four Pillars for Improving the Quality of Safety-Critical Software-Reliant Systems," April 2013. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=47791.
3. Peter Feiler, Charles Weinstock, John Goodenough, Julien Delange, Ari Klein, and Neil Ernst, "Improving Quality Using Architecture Fault Analysis with Confidence Arguments," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2015-TR-006, 2015.
4. Bertin, V.; Closse, E.; Poize, M.; Pulou, J.; Sifakis, J.; Venier, P.; Weil, D.; Yovine, S., "TAXYS=Esterel+Kronos. A tool for verifying real-time properties of embedded systems," Proceedings of the 40th IEEE Conference on Decision and Control, 2001.
5. Sagar Chaki, Arie Gurfinkel, Ofer Strichman, Time-Bounded Analysis of Real-Time Systems, Proceedings of Formal Methods in Computer-Aided Design (FMCAD), 2011.
6. Brian R. Larson, Patrice Chalin, John Hatcliff: "BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software." Proceedings of the 2013 NASA Formal Methods Conference, pp. 276-290.
7. ”SAE Architecture Analysis and Design Language (AADL) Annex Volume 2, Annex D: Behavior Model Annex”. SAE standard AS5506/2. SAE International, 2011.
8. Julien Delange, Peter Feiler. Incremental Latency Analysis of Heterogeneous Cyber-Physical Systems. In Real-Time and Distributed Computing in Emerging Applications (REACTION) 2014.
9. ”SAE Architecture Analysis and Design Language (AADL) Annex Volume 1A, Annex E: Error Model V2 Annex”. SAE standard AS5506/1A. SAE International, 2015.

10. Julien Delange, Peter H. Feiler, Architecture Fault Modeling with the AADL Error-Model Annex. 40th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2014).
11. Julien Delange and Peter Feiler, Supporting the ARP4761 Safety Assessment Process with AADL Proceedings of Embedded Real-Time Software and Systems 2014 (ERTSS2014), Toulouse, France. February 2014.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Benefits of Model Based System Engineering for Avionics Systems

Thierry Le Sergent, François-Xavier Dormoy, Alain Le Guennec (Esterel Technologies)
thierry.lesergent@esterel-technologies.com
francois-xavier.dormoy @esterel-technologies.com
alain.leguennec@esterel-technologies.com

9, rue Michel Labrousse,
31100, Toulouse, France

Keywords: Model Based System Engineering, Software Engineering, SysML, Domain Specific Language, SCADE, Avionics System, Aircraft Braking System

1. Introduction

Avionic system design is an extremely complex activity due to:

- Different concerns from different teams:
    - Complexity of functions to realize
    - Hardware and software redundancies to manage safety considerations
    - Communication protocols
    - Complex hardware network with configuration switches,
    - Etc.
- Amount of data to manage: 1,000s of data
- Resource sharing principle (CPU, network, …)

Current methods to manage system engineering complexity rely on refinement levels and view-points. Industries promote ICDs (Interface Control Documents) as a contractual means to assign the development of system components. These are basically large databases that gather the result of system architecture design.

In the domain of safety critical software, the benefits of a model based approach are well established. Now, using an example of avionics architecture, we will show how Model Based System Engineering (MBSE) brings many benefits when the tools are suited for system designers and applied to solve industrial challenges.

Proponents of MBSE provide strong arguments for improving the efficiency of complex system development, promising to achieve at least the same level of quality in a shorter time. However, this gain in efficiency must be proven. This paper details the way a realistic complex avionics system can be designed efficiently using a MBSE tool, with hundreds of data passed through ARINC 429 and ARINC 664-P7 messages.

A classical methodology is followed to manage the different levels of concern:

- Functional architecture
    - Functional data flow communications
- Software architecture
    - Function organization into software components
    - Software message definition and propagation between the components
- Platform hardware architecture
    - Definition of buses, switches, and computing units
    - Software to hardware mapping
    - Virtual link definition for the ARINC 664-P7 communication
- Software design, code generation and verification with SCADE Suite [10]

In order to sustain industrial deployment as a cost-effective MBSE tool, the following must be addressed:

- Interoperability – By conforming to a standard, such as SysML [6], users avoid the risk of vendor lock-in and can integrate with other technologies.
- Usability – In order to overcome reluctance to change, the tool should speak the language of system engineers rather than forcing engineers to force their design into an awkward formalism.
- Efficiency - Tools and methodology should bring savings in design, analysis, and document generation

The SCADE System Avionics Package delivers the following answers:

- Fully customizable interface driven by Domain Specific Languages
- Clean separation and consistent relationships of the Functional, Software and Hardware layers
- Templates for standard avionics protocols (ARINC 429, ARINC 664-P7, CAN provided)
- Intuitive hierarchical data modeling and automated ICD generation

These means are detailed in this paper; the design of the industrial case study that is first introduced has demonstrated the efficiency of the tool support.

2. Avionics system case study

For this case study, Dassault Aviation provided a simplified representative example of a Braking System, based on a COM-MON architecture. Principles of the COM-MON design are not detailed but focus is made on the ARINC 664-P7 and ARINC 429 communications between the COM, MON and other systems.

- At the functional level, COM and MON interact with 9 sub-systems. 175 functional data are considered.
- At the software level, COM and MON partitions are respectively interacting with 12 partitions. 14 ARINC 664-P7 messages and 48 A429 messages are defined.
- The platform level contains 4 Processing cores (CPU), 4 Switches, a dual A/B ARINC 664-P7 network, and 30 Virtual Links (VL).

Based on this example, we realized a complete retro-fit in SCADE System through the following steps:

- Description of the entire architecture (Functional, Software and Platform)
- Production of the application ICDs
- Management of data consistency checks across the complete application
- Management of Virtual Link paths to facilitate switch configurations
- Analysis of message allocation with respect to network bandwidth
- Synchronization of the system architecture with the software design

Screen shots and details of the complete example are provided after we have introduced all the tool features that enable efficient support of the design method.

3. Proposed Model Based Engineering Method and Tool Support

As introduced in the first section, a traditional system design approach is used, relying on several abstraction layers: functional, software (sometimes called “logical”), and platform. Our approach is unique in how it defines and manages the relations between these levels of abstraction.

The following technical means are hereafter detailed:

- Allocation and interface specification, relying on data management
- Customization of SysML objects
- Analysis of the model, in particular to produce the Interface Control Documents (ICD).

3.1. Allocation and interface specification

When designing a system in several abstracting layers, consistency and completeness of each layer must be ensured. The difficulty comes from the fact that the abstraction levels are not a simple hierarchical decomposition. The functional, software, and hardware views are all hierarchical, but with a different hierarchy.

The relationship between the different layers is called projection. SysML comes with a dedicated construct, allocations, to support this. Projections provide a straightforward way to show the realization of a function by a software component, and show how a software component is run by a hardware component.

Before focusing on the interface allocations themselves, it should be noted that the intention of a projection is to specify precisely which item shall be supported by which item. The projection must take into account that blocks may be instantiated several times. The difficulty arising from the hierarchical instantiation is solved in SCADE System though a block replica mechanism detailed in [13]. The result of the replication mechanism is that each object from the real world is represented with a dedicated object in the model. This replication mechanism is reused for the data objects presented below.

Projection of the component interfaces can be realized in the same simple way as for block components. But this does not lighten the burden of making all interfaces consistent with each other. Let’s consider the following case:

- A function F produces data D, for several other functions defined in the functional decomposition.
- Each function is allocated to a software component, some sharing the same component, others not.

- The software components exchange messages in order to transmit the functional data.

It is easy to specify that the software component S supporting F shall produce a message M carrying D. But what about the other sides of the communication? The message M must be sent to all software components that support a function receiving data D. This rule must be maintained whatever the data propagation and function allocations, something that may evolve during the design process.

Another consideration is that ARINC 429 and ARINC 664-P7 messages can carry several data elements. The designer may define either one message carrying several data to other software components, even if some data are not required by all the message recipients, or to define several messages. All these challenges are supported by a concept of data.

3.2. Data management

SCADE System comes with an original and powerful means to specify the interface allocations efficiently. It relies on data objects that can be structured and propagated.

The data propagation mechanism first published in [13] is summarized here:

- Data are implemented as SysML blocks, allowing SCADE System to conform to the SysML meta-model, but managed as data in the tool.
- Data are propagated along block ports and connectors thanks to a dedicated feature of the tool
- Propagation of a data D out of a block B through one of its ports consists in the creation of a data proxy in the parent blocks instantiating B.
- All proxies are connected as a chain; the tool allows managing the data attributes (feature detailed in section 3.3) whatever the selection of one proxy of the chain.

This mechanism brings two important benefits. First, it allows handling a complete “path” through a block hierarchy and connections from a single data element. Second, it allows managing thousands of data, each broadcasted or multi-casted in a straightforward way by the designer, without graphical scalability issues by avoiding the creation of numerous ports and connectors (the proxies are automatically created from path selection).

Individual data are structured thanks to an internal model transformation:

- Data structures are defined with classical structured types
- Applying a structured type to data replicates the structure inside of the data definition. This allows for each data to have its own individual fields.

Structured data is an ideal means to model messages; the message fields represent place-holders for the functional data. Simple allocations of functional data (defined at the functional abstraction layer) to a field of a structured data message represents the fact the message carry the functional at a certain place in the message. This would not be possible without the internal replication of the message type because the field of type would be shared by all the messages sharing the same type.

Allocating a propagated functional data to a propagated message data implements the interface allocations between the abstraction layers.

An additional feature allows managing the user interface scalability: There may be several thousand of functional data, and data messages in a model. Managing the allocation between the different levels of abstraction is challenging for the usability perspective of a graphical tool. Even a traditional list of objects can be cumbersome for the users. SCADE System IDE comes with a customizable filtering mechanism that restricts possible candidates to correct allocations and propagation. The principle is the following:

- A checking rule verifies that a functional data FD propagated from function FA, allocated to Component CA, to function FB, allocated to component CB, is allocated to message data MD that goes from CA to CB. This check takes into account both the possible multiple propagation target of a data, and possible multiple allocations to messages to handle redundancies.
- In addition to automated verification and report generation at the end of the design, the checking rule is used dynamically by the UI to filter out the allocation and propagation possibilities that would lead to a violation of the rule. That way, the UI exposes a much shorter list of possible selections, and correct designs are realized much faster.

This mechanism extends very easily: users can program their own verification rules from the model API, and use them both for final batch verifications, and as filters in the allocation and the data propagation interfaces.

3.3. DSL vs. SysML: Configuration of the SCADE System tool

The method presented above only requires blocks, ports, connectors, data, and structured type objects. However, these constructs fall short of easily addressing a specific domain. The ability of a tool to meet the needs of domain expert engineers is a key success deployment factor, which must go beyond these few generic SysML objects.

There are traditionally two competing approaches when it comes to domain-specific system modeling:

- “Pure” Domain-Specific Languages (DSLs)
- Customizations of generic modeling solutions, e.g. UML with profiles such as SysML

Proponents of the first approach appreciate the total freedom that DSLs provide in order to best fit actual domain concerns, without incurring the cost of supporting legacy UML peculiarities. Proponents of the second approach claim that reliance on standards such as UML provide better long term sustainability and interoperability.

SCADE System stands out with an original hybrid approach that combines the best of both worlds: While the internals of SCADE System rely on UML/SysML and profiling (therefore providing full standard support and interoperability), SCADE System still presents itself as a domain-agnostic system modeling language and toolset, via a pure DSL “virtual layer” implemented transparently on top of UML and SysML. This layer is naturally extended to accommodate domain-specific needs without starting from scratch: The domain specialist simply designs a domain-specific language as a meta-model that specializes SCADE System’s domain-agnostic meta-model. Additional information is conveyed by appropriate attributes in specializing meta-classes and relationships among them; additional constraints imposed by the domain are enforced by redefinitions of existing relationships.

SCADE System transparently handles the intricacies of UML profile management and dynamically adjusts its interface to the domain-specific “configuration”, offering new modeling constructs (in toolbars, property pages, etc.) while forbidding others according to constraints of the domain. Moreover, models based on a domain configuration can be manipulated via a “pure” API derived from the domain meta-model, not UML concepts and stereotypes.

Figure 1 below shows the tool workflow in practice: the domain specialist designs the domain meta-model; from it, a “configuration plug-in” is automatically generated. By loading this plug-in file, SCADE System is transformed into a Domain Specific tool with the customized “creation palette” and object properties.

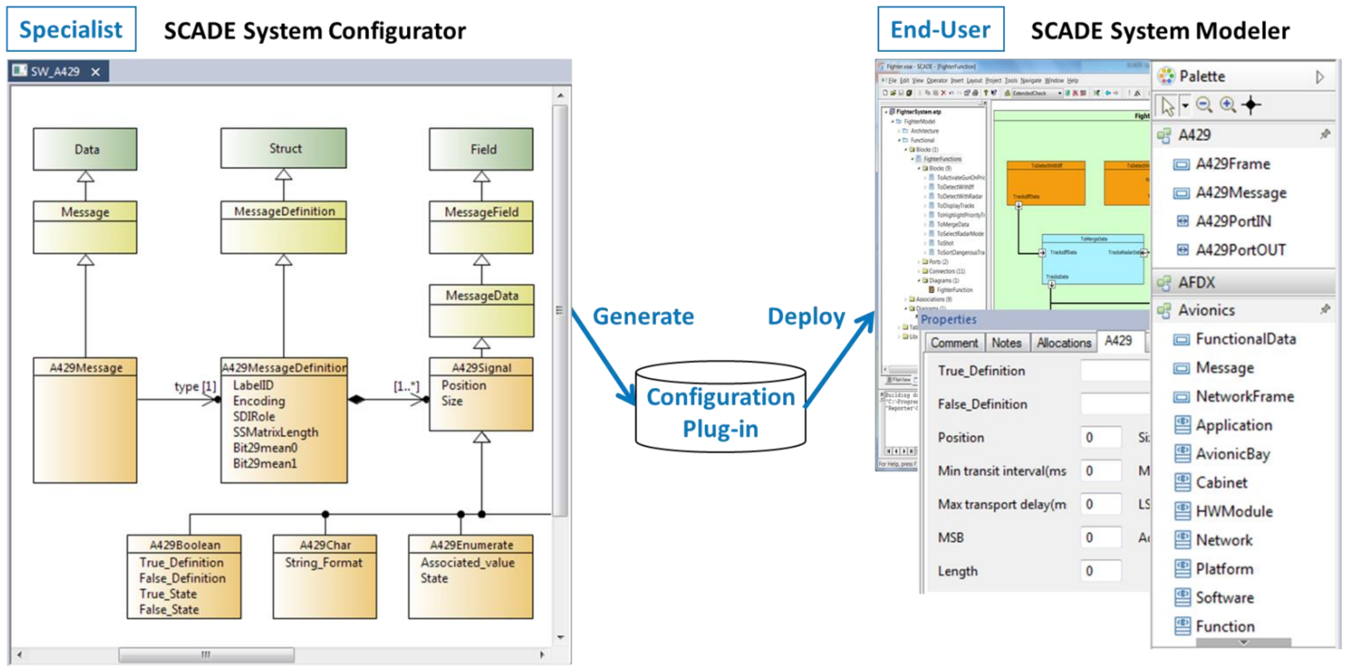


Figure 1: System Configuration Workflow

Figure 1 also shows an extract of the ARINC 429 meta-model.

- At the top of the diagram, Data, Struct and Field refer to SCADE System meta model;
- Below, Message, MessageDefinition, MessageField and MessageData are defined in the “generic” Avionics configuration. They all inherit from SCADE System meta classes
- Finally at the bottom of the diagram, the ARINC 429 meta-classes refine the Avionics meta-classes, providing all information needed for the definition of ARINC 429 messages.

Similar meta-models are provided for ARINC 664-P7 and CAN messages. The same technique is used to define all meta-classes needed at the platform definition layer, with the physical ports, switches, etc.

3.4. *ICD production*

Now that we have described the data management process, Interface Control Documents can be generated from the model. These Excel sheets are needed by many different development teams, each requiring different information, presented in the way they expect. For example:

- Partition table
- Messages definition with their parameters
- Messages source and targets with ARINC 664-P7 ports, transmission rate, length, etc.
- ARINC 664-P7 Virtual Links definition

Each table being specialized for a specific usage, the tool should not rely on dedicated built-in tables, but offer a means for each user to customize their own view. SCADE System comes with a simple but powerful means:

- A dedicated dialog allows defining the hierarchy of objects to display in a table: the list of possible children, references, or even arbitrary queries is proposed for selection.
- For each line in this tree, the list of possible attributes, references, or arbitrary queries is proposed to create columns in the table.

These lists of possible children, references and attributes are directly exposed from the model API. This includes the domain meta-model API specified. So, in a few clicks one can for example set-up the table of all ARINC 429 messages, with their target and with the parameters from data allocated from the functional design.

All table examples listed below have been produced for the designed aircraft braking system. They comply with the original MS Excel tables that were given at the start of the project. Of course the displayed tables can be exported in a click to MS Excel files.

4. Application to an Avionics system case study

We will now apply the methodology and tools to the avionics system case study described at the beginning of the paper. As explained in section 3 we proceed in layers focusing on data management (producing and consuming).

4.1. *Functional architecture*

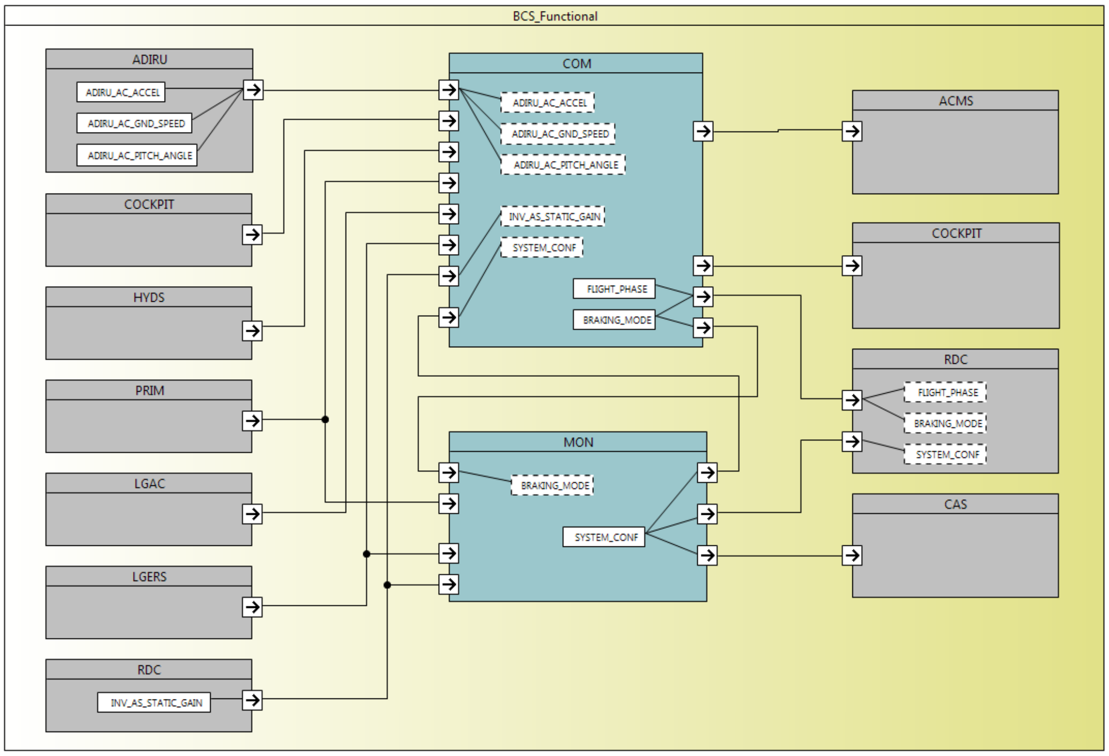
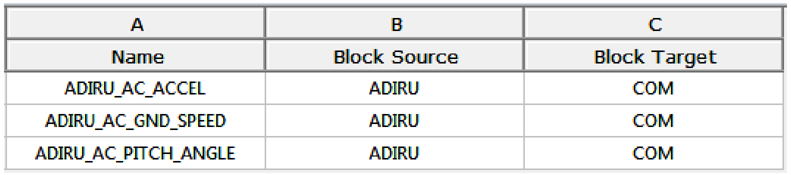


Figure 2: Functional architecture

Beginning with the functional architecture, we define the functions of the Braking System and the data produced and consumed (required) for each function.

The COM function of the Braking System needs 3 functional data produced by the ADIRU Function (ADIRU_AC_ACCEL: Aircraft Acceleration; ADIRU_AC_GND_SPEED: Aircraft Ground Speed; ADIRU_AC_PITCH_ANGLE: Aircraft Pitch Angle). The graphical diagram shows the functions and their data flow dependencies. Designers have fine-grained control over the functional data displayed in the diagram.

As a result of this stage, Functional Data with producer/consumer tables are available for each Function as illustrated in Figure 4 below for the ADIRU and a global table with all functional data produced and consumed by the Braking System application. Automated checks allows immediate detection of functional data that are not produced or functional data that are not consumed (useless data).



A	B	C
Name	Block Source	Block Target
ADIRU_AC_ACCEL	ADIRU	COM
ADIRU_AC_GND_SPEED	ADIRU	COM
ADIRU_AC_PITCH_ANGLE	ADIRU	COM

Figure 3: ADIRU functional data table

4.2. Software architecture

The software architecture defines the software components of the Braking System and the messages produced and consumed (required) by each software component. Again the messages are propagated in the software architecture by the data propagation mechanism.

In the example illustrated below, the BCS_COM software component of the Braking System requires one message (MSG_ADIRU_COM_C10: ARINC 667-P7 message) produced by the ADIRU component.

The Software architect may use several diagrams to focus on ARINC 664-P7 or ARINC 429 communication as illustrated in figures 5 and 6 below.

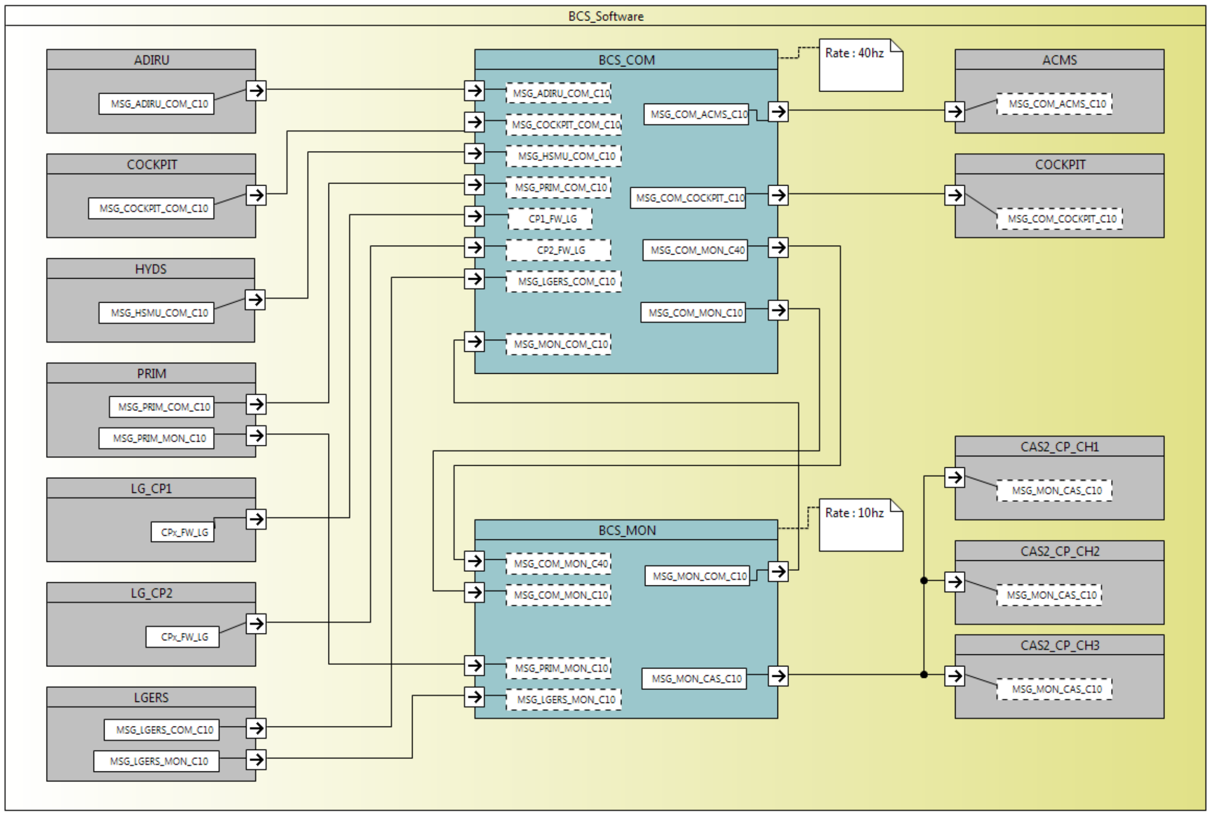


Figure 4: Software architecture focusing on ARINC 664-P7 Communication

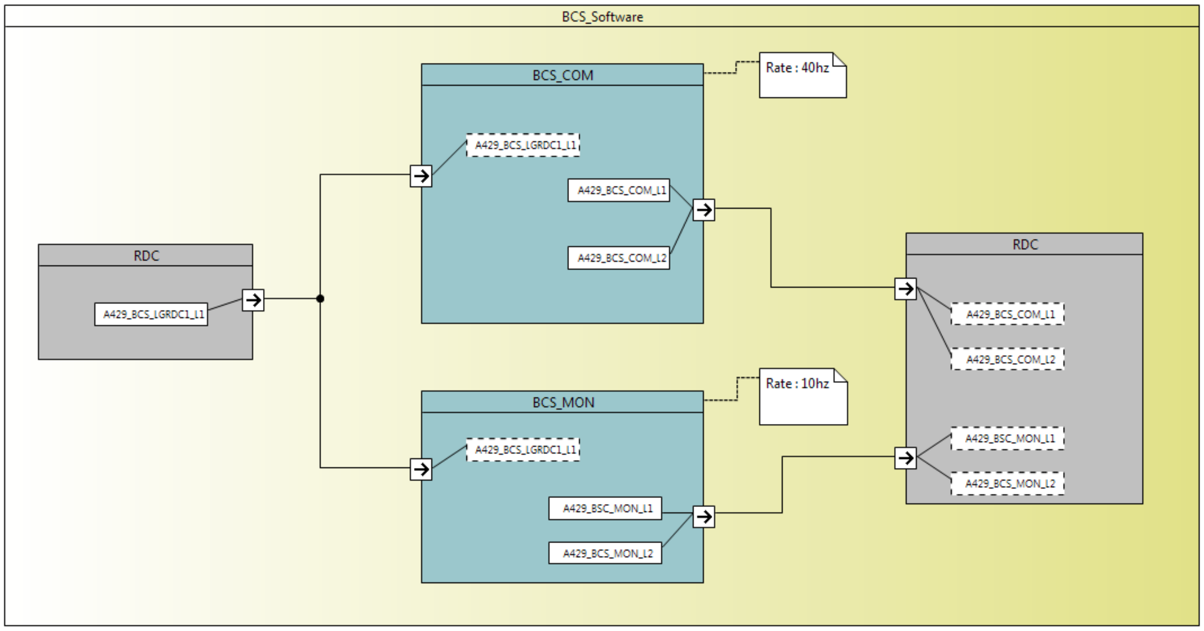
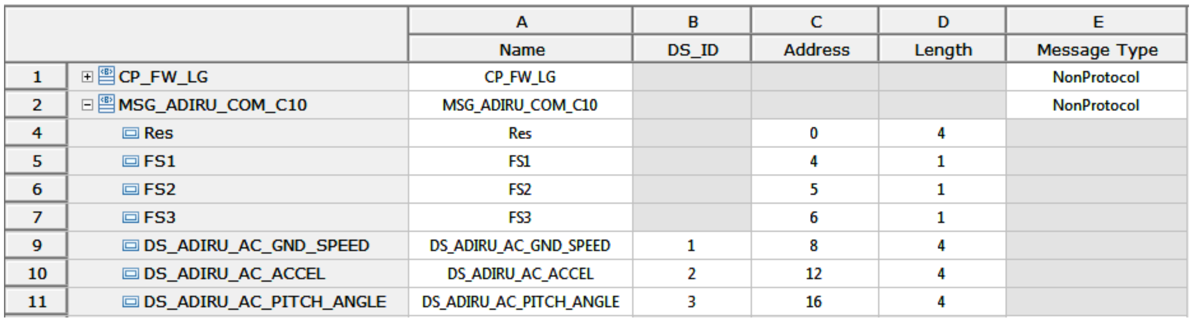


Figure 5: Software architecture focusing on ARINC 429 Communication

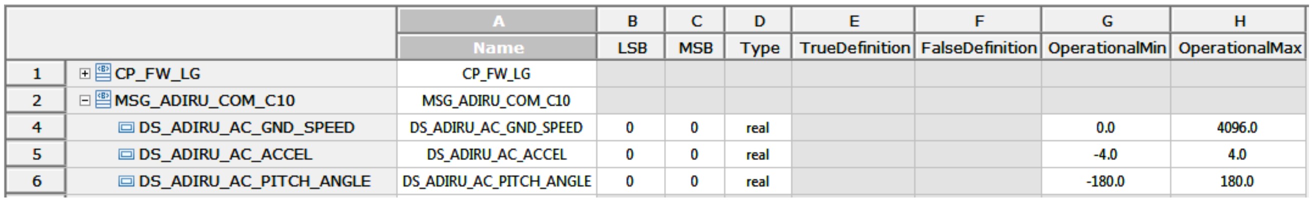
At the software level we also describe all messages (ARINC 664-P7 and ARINC 429). Figure 7 illustrates the MSG_ADIRU_COM_C10 (ARINC 664-P7) message, with its three data set containing the functional data AC_GND_SPEED, AC_ACCEL and AC_PITCH_ANGLE.



		A	B	C	D	E
		Name	DS_ID	Address	Length	Message Type
1	⊞ CP_FW_LG	CP_FW_LG				NonProtocol
2	⊟ MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10				NonProtocol
4	Res	Res		0	4	
5	FS1	FS1		4	1	
6	FS2	FS2		5	1	
7	FS3	FS3		6	1	
9	DS_ADIRU_AC_GND_SPEED	DS_ADIRU_AC_GND_SPEED	1	8	4	
10	DS_ADIRU_AC_ACCEL	DS_ADIRU_AC_ACCEL	2	12	4	
11	DS_ADIRU_AC_PITCH_ANGLE	DS_ADIRU_AC_PITCH_ANGLE	3	16	4	

Figure 6: ARINC 664-P7 Message definition Table

Data Set information are gathered in another Table as illustrated below in figure 8.

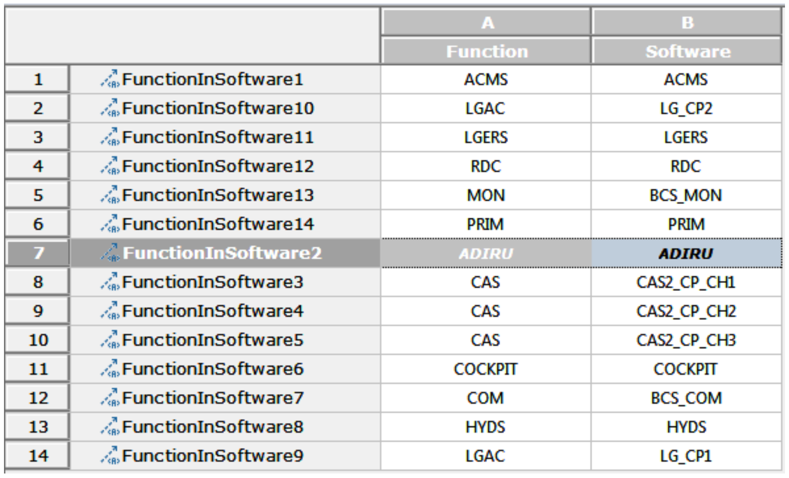


		A	B	C	D	E	F	G	H
		Name	LSB	MSB	Type	TrueDefinition	FalseDefinition	OperationalMin	OperationalMax
1	⊞ CP_FW_LG	CP_FW_LG							
2	⊟ MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10							
4	DS_ADIRU_AC_GND_SPEED	DS_ADIRU_AC_GND_SPEED	0	0	real			0.0	4096.0
5	DS_ADIRU_AC_ACCEL	DS_ADIRU_AC_ACCEL	0	0	real			-4.0	4.0
6	DS_ADIRU_AC_PITCH_ANGLE	DS_ADIRU_AC_PITCH_ANGLE	0	0	real			-180.0	180.0

Figure 7: ARINC 664-P7 Data Set Table

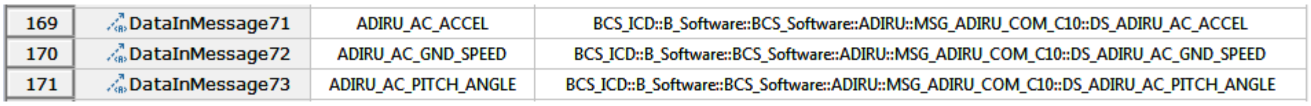
4.3. Functional to Software mapping

Now that we have defined the functional and the software architecture, the functional to software mapping consists in the allocations of functions to the software components (illustrated in figure 9), and the functional data mapping to software messages (illustrated in figure 10). Allocation tables are used for that purpose.



		A	B
		Function	Software
1	FunctionInSoftware1	ACMS	ACMS
2	FunctionInSoftware10	LGAC	LG_CP2
3	FunctionInSoftware11	LGERS	LGERS
4	FunctionInSoftware12	RDC	RDC
5	FunctionInSoftware13	MON	BCS_MON
6	FunctionInSoftware14	PRIM	PRIM
7	FunctionInSoftware2	ADIRU	ADIRU
8	FunctionInSoftware3	CAS	CAS2_CP_CH1
9	FunctionInSoftware4	CAS	CAS2_CP_CH2
10	FunctionInSoftware5	CAS	CAS2_CP_CH3
11	FunctionInSoftware6	COCKPIT	COCKPIT
12	FunctionInSoftware7	COM	BCS_COM
13	FunctionInSoftware8	HYDS	HYDS
14	FunctionInSoftware9	LGAC	LG_CP1

Figure 8: Function to Software allocation Table



169	DataInMessage71	ADIRU_AC_ACCEL	BCS_ICD::B_Software::BCS_Software::ADIRU::MSG_ADIRU_COM_C10::DS_ADIRU_AC_ACCEL
170	DataInMessage72	ADIRU_AC_GND_SPEED	BCS_ICD::B_Software::BCS_Software::ADIRU::MSG_ADIRU_COM_C10::DS_ADIRU_AC_GND_SPEED
171	DataInMessage73	ADIRU_AC_PITCH_ANGLE	BCS_ICD::B_Software::BCS_Software::ADIRU::MSG_ADIRU_COM_C10::DS_ADIRU_AC_PITCH_ANGLE

Figure 9: Extract of Functional Data to Software Message allocation Table

Automated consistency checks confirm that these allocations are consistent with the data and message propagation: the message to which a functional data element is allocated to must be produced by a software component to which the producer function is allocated to, and must be propagated to the software components to which the consumer function is allocated to. Inefficiencies are also detected, for example when a message carries a data not needed by all recipients of the message. One can now analyze in detail the design tradeoff for message definitions.

4.4. Platform architecture

At this stage we capture the platform components (CPUs, switches, buses, external devices, etc) describing the possible resources (CPU, memory, bandwidth, etc) available to the architect to deploy the application. The BCS platform consists of 4 CPUs, 4 Switches, an external device (RDC) and several buses as described in Figure 11.

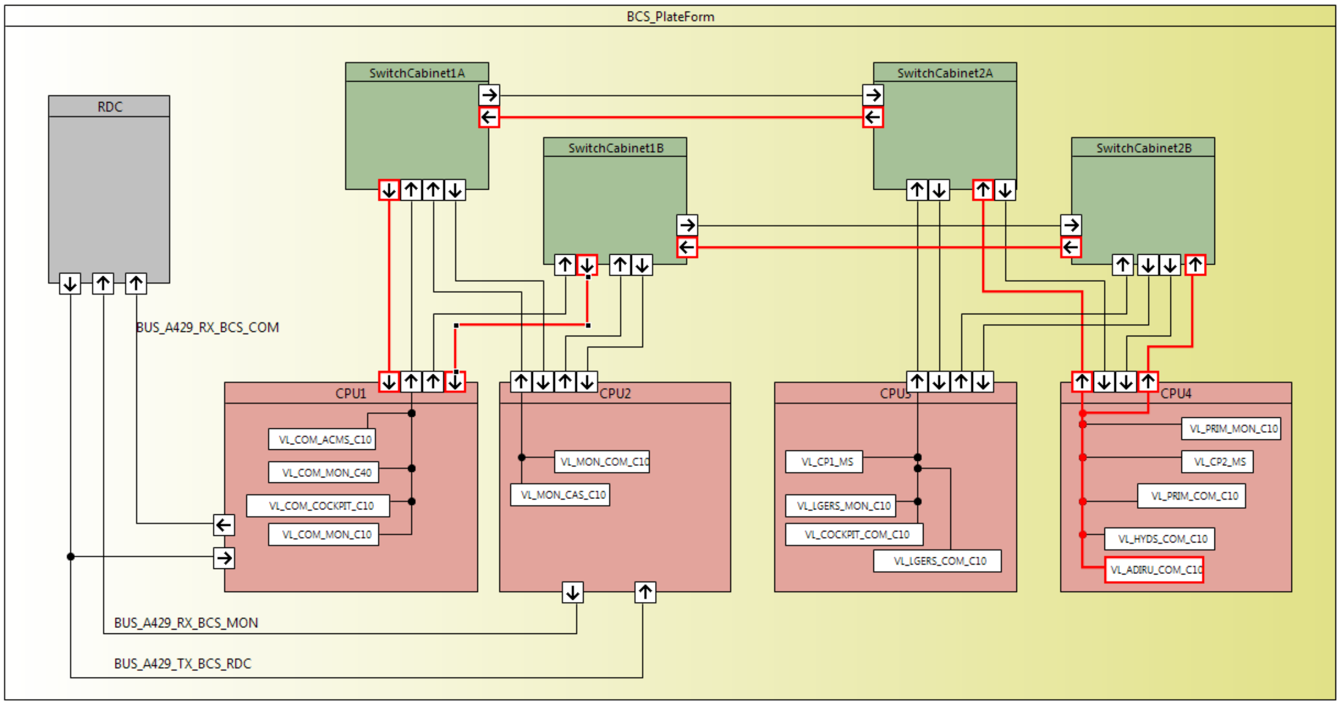


Figure 10: Platform architecture illustrating Chanel A, Chanel B Virtual link routes

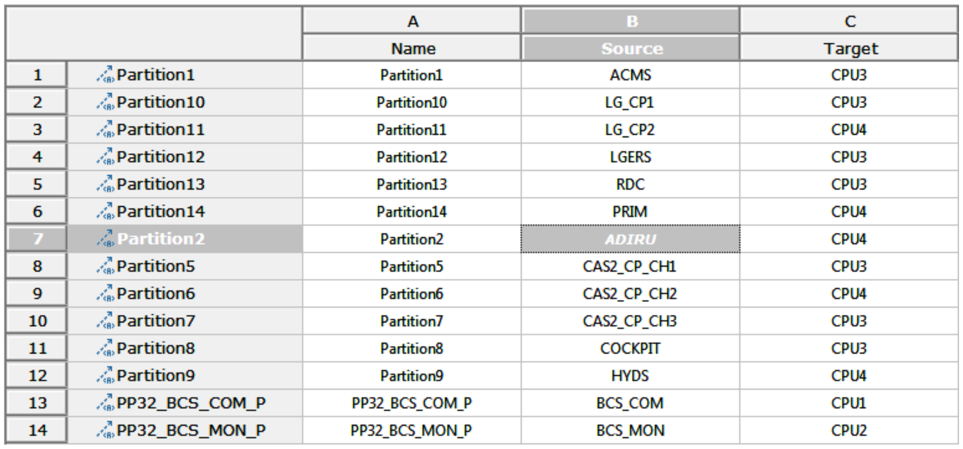
At this platform level, we allocate software components to the CPUs and define message paths between them. For ARINC 664-P7, these paths are known as Virtual Links. Again, these paths are implemented as dedicated data propagated in this architecture. For example, with the ADIRU software being hosted in CPU4 and the COM being hosted in CPU1, the VL associated to MSG_ADIRU_COM_C10 are defined for channel A and channel B, following the classical redundancy means in ARINC 664-P7 communication.

Virtual Links routing are automatically produced in Tables enabling configuration of switches.

4.5. *Software to Platform mapping*

One of the important tasks for the integrator is to allocate the software components onto the platform and optimize usage of available resources. This is one of the main challenges for IMA (Integrated Modular Avionics).

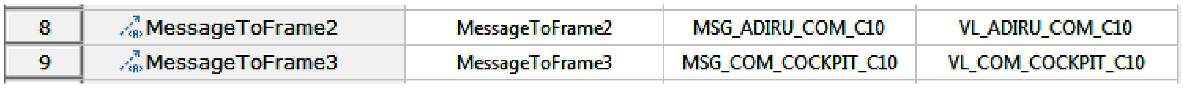
Allocation of the software components (Partition) to CPUs is illustrated in Figure 12 below.



		A	B	C
		Name	Source	Target
1	Partition1	Partition1	ACMS	CPU3
2	Partition10	Partition10	LG_CP1	CPU3
3	Partition11	Partition11	LG_CP2	CPU4
4	Partition12	Partition12	LGERS	CPU3
5	Partition13	Partition13	RDC	CPU3
6	Partition14	Partition14	PRIM	CPU4
7	Partition2	Partition2	ADIRU	CPU4
8	Partition5	Partition5	CAS2_CP_CH1	CPU3
9	Partition6	Partition6	CAS2_CP_CH2	CPU4
10	Partition7	Partition7	CAS2_CP_CH3	CPU3
11	Partition8	Partition8	COCKPIT	CPU3
12	Partition9	Partition9	HYDS	CPU4
13	PP32_BCS_COM_P	PP32_BCS_COM_P	BCS_COM	CPU1
14	PP32_BCS_MON_P	PP32_BCS_MON_P	BCS_MON	CPU2

Figure 11: Software (Partition) to Platform projection

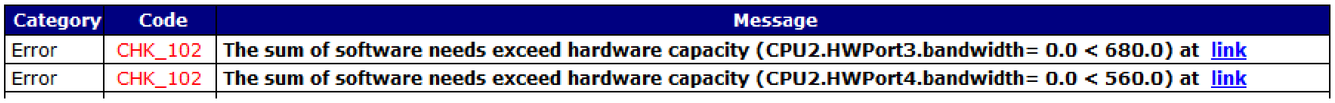
In the same way software messages allocated to Virtual Links are illustrated in table 13 below.



8	MessageToFrame2	MessageToFrame2	MSG_ADIRU_COM_C10	VL_ADIRU_COM_C10
9	MessageToFrame3	MessageToFrame3	MSG_COM_COCKPIT_C10	VL_COM_COCKPIT_C10

Figure 12: Extract of Message to Virtual Link allocation

We can then perform resource usage checks, such as bandwidth checks, as illustrated in Figure 14 below.

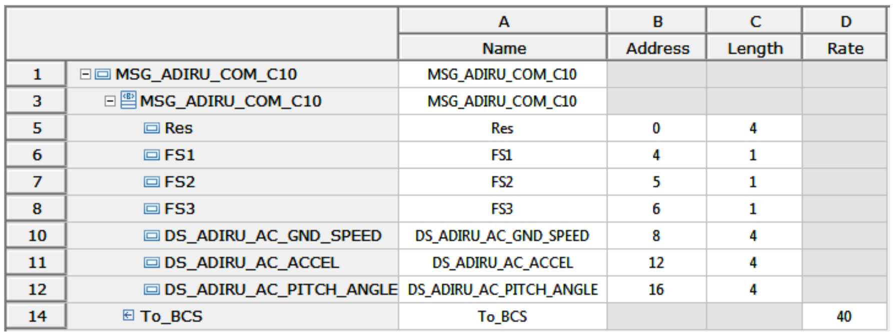


Category	Code	Message
Error	CHK_102	**The sum of software needs exceed hardware capacity (CPU2.HWPort3.bandwidth= 0.0 < 680.0) at** link
Error	CHK_102	**The sum of software needs exceed hardware capacity (CPU2.HWPort4.bandwidth= 0.0 < 560.0) at** link

Figure 13: ARINC 664-P7 bandwidth checks results

4.6. *Braking System ICD (Interface Communication Document)*

Thanks to this final allocation step we have a complete description of data communication from the functional level down to platform integration from which ICD tables are automatically produced as illustrated in Figure 15 below.



		A	B	C	D
		Name	Address	Length	Rate
1	MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10			
3	MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10			
5	Res	Res	0	4	
6	FS1	FS1	4	1	
7	FS2	FS2	5	1	
8	FS3	FS3	6	1	
10	DS_ADIRU_AC_GND_SPEED	DS_ADIRU_AC_GND_SPEED	8	4	
11	DS_ADIRU_AC_ACCEL	DS_ADIRU_AC_ACCEL	12	4	
12	DS_ADIRU_AC_PITCH_ANGLE	DS_ADIRU_AC_PITCH_ANGLE	16	4	
14	To_BCS	To_BCS			40

Figure 14: Extract of Braking System ICD

5. Comparison with other approaches

The NASA handbook [4] states that a clean process must be set up to "*identify and resolve interface incompatibilities and to determine the impact of interface design changes*". Management of Interface Control Documents (ICDs) [5] is indeed at the center of most industries' system engineering processes, but most often supported by tedious manual processes based on MS Excel files cross analysed and reviewed by the different engineering teams involved.

More robust processes rely on databases. They provide scalability, and the database schema enforces some design rules. However, this is not as powerful as the meta-modelling capability provided by UML-based tools, and does not support a graphical representation which is an essential communication means between engineers.

SysML [6] is the most known standard in system engineering, but it does not come with specific constructs to model software messages that are exhanged between "blocks". SCADE System Avionics is compliant with SysML, and provides such constructs. Other standards have been set-up which focus on the component interfaces. We now compare our approch with three of them that have been deployed in the industry.

EAST-ADL [7], initially defined in the European ITEA EAST-EEA project that has been then aligned with the AUTOSAR automotive standard was a good source of inspiration. In the same way as the method presented here, the central features of EAST-ADL, is its multi-layer approach, which defines multiple abstraction levels and distributes all development information across these levels. Just like the SCADE System Avionics package, EAST-ADL is supported by a UML Profile. The main difference is in the focus we have put on the detailed definition of the component interfaces that gather information from the different levels of abstraction. On the other hand EAST-ADL provides a focus on timing information that has not been yet considered in the SCADE System Avionics Package.

The SAE "Architecture and Analysis Description Language" (AADL) [8], has its roots in the avionics domain. AADL allows for the description of both software and hardware parts of a system. In contrast to SysML's limited generic components of "block", "port", etc., AADL provides precise modeling concepts to describe the runtime architecture of application systems in terms of concurrent tasks, their interactions, and their mapping onto an execution platform. It also separates the definition of block interfaces from their implementation, but does not make the links between functional, software and platform interfaces in the way presented in this paper.

The more recent FACE standard [9] also focuses on the non-ambiguous specification of interfaces for "Unit Of Portability" (UoP). It describes the component interfaces at three different layers, conceptual (e.g. mass, length …), logical units (e.g. Kg, millimeters …), and platform implementation (e.g. uint8, float64 …). This complements very well our approach. Indeed it allows setting a "meaning" to the functional data used at the top level of the process we have presented, while FACE itself does not detail the way this information shall be carried into software messages.

It is perfectly possible to define with SCADE System Configurator the EAST-ADL, the AADL, and the FACE meta-models. On-going work to support the FACE standard as a complement of the Avionics package presented here will be published in the coming months.

6. Conclusion

This paper presents a model based development approach and tooling enabling efficient and agile process development focusing on data. Applied to a case study based on a braking system example, we have demonstrated a complete flow from functional architecture capture down to platform deployment showing the main benefits of a model based approach:

- Automated checks ensuring that the functional, software, and platform architecture are consistent with each other
- Data flow checks ensuring proper usage and production of data all along the process
- Platform resource and usage domain checks enabling design tradeoff in platform definition
- Automatic generation of ICD (Interface Communication Document)
- Automatic generation of configuration files (for OS and switches)
- Full description of communication (CAN, ARINC 429, ARINC 664-P7, discrete)
- Full description of Software architecture

Other important topics covered at tool and methodology level are not addressed in this paper:

- Requirement traceability all along the process
- PLM/ALM (Product or Application Lifecycle Management) linkage
- Product Line Engineering and Variant Management

- Synchronization of System and Software engineers work

Starting from the provided ICD files of the braking system example, the complete retro-fit in SCADE System has been achieved in a couple of months. Thanks to the model based solution, with a user interface that speaks the language of the avionics system designer, design modification and regeneration of consistent ICDs is now mature. This demonstrates a large efficiency and the capability to significantly accelerate the development cycle in handling design changes.

Well supported by tools with features detailed in this paper, we believe the system design methodology can reach the maturity level the software component design has achieved since several years.

7. Acknowledgment

We want to thank Mickael Lafaye and Jean-Pascal Rotteleur from Dassault Aviation for their help and the fruitful inputs they provided during the Avionics Case Study

8. References

1. “Systems Engineering Handbook, a Guide for System Life Cycle Processes and Activities”, SE Handbook Working Group, INCOSE, January 2010.
2. “ARP4754 Rev A. Guidelines for Development of Civil Aircraft and Systems”, SAE Aerospace, Revised 2010-12
3. “DO 297/ED124 IMA, INTEGRATED MODULAR AVIONICS (IMA) DEVELOPMENT GUIDANCE AND CERTIFICATION CONSIDERATIONS”, RTCA - 2005
4. “NASA Systems Engineering Handbook”, NASA/SP-2007-6105 Rev1
5. “Training Manual for Elements of Interface Definition and Control”. Vincent R. Lalli, Robert E. Kastner, and Henry N. Hartt, NASA Reference Publication 1370, January 1997.
6. “OMG Systems Modeling Language (OMG SysML)”, OMG, Version 1.2, June 2010
7. EAST-ADL, http://www.east-adl
8. “Architecture Analysis & Design Language (AADL)”, SAE Aerospace AS5506B. http://standards.sae.org/as5506b/
9. “Future Airborn Capability Environmennt (FACE)”. http://www.opengroup.org/face.
10. Esterel technologies SCADE products, http://www.esterel-technologies.com
11. “SCADE System, a comprehensive toolset for smooth transition from Model-Based System Engineering to certified embedded control and display software”, Thierry Le Sergent, Alain Le Guennec, François Terrier, Yann Tanguy, Sébastien Gérard. ERTS 2012
12. “Integrating System and Software Engineering Activities for Integrated Modular Avionics Applications”, Thierry Le Sergent, Frederic Roméas, Olivier Tourillion. 2012 SAE Aerospace Electronics and Avionics Systems Conference, Phoenix Arizona, USA.
13. “Data Based System Engineering: ICDs management with SysML”, Thierry Le Sergent, Alain Le Guennec. ERTS 2014

A Seamless Model-Transformation between System and Software Development Tools

Georg Macher*†, Harald Sporer*, Eric Armengaud†, Eugen Brenner* and Christian Kreiner*

*Institute for Technical Informatics, Graz University of Technology, AUSTRIA
Email: {georg.macher, sporer, brenner, christian.kreiner}@tugraz.at

†AVL List GmbH, Graz, AUSTRIA
Email: {georg.macher, eric.armengaud}@avl.com

***Abstract*—The development of dependable embedded automotive systems faces many challenges arising from increasing complexity, coexistence of critical and non-critical applications, and the emergence of new architectural paradigms on the one hand, to short time-to-market intervals on the other hand. This situation requires tools to improve efficiency and consistence of development models along the entire development lifecycle. The existing solutions to date are still all too frequently insufficient when transforming system models with higher levels of abstraction to more concrete engineering models (such as software engineering models). Future automotive systems require appropriate structuring and abstraction in terms of modularization, separation of concerns, and supporting interactions between system, and component development.**

However, refinement of system designs into hardware and software implementations is still a tedious task. The aim of this work is to enhance an automotive model-driven system engineering framework with software-architecture design capabilities and a model-transformation framework to enable a seamless description of safety-critical systems, from requirements at the system level down to software component implementation in a bidirectional manner.

Keywords*—*Automotive, model-based development, reuse, traceability, model-based software engineering, ISO 26262.

I. Introduction

Embedded systems are already integrated in our everyday lives and play a central role in all domains including automotive, aerospace, healthcare, manufacturing industry, the energy sector, or consumer electronics. In 2010, the embedded systems market accounted for almost 852 billion dollars worldwide, and is expected to reach 1.5 trillion by 2015 (assuming an annual growth rate of 12%) [18]. Current premium cars implement more than 90 electronic control units (ECU) per car with close to 1 Gigabyte software code [6], these are responsible for 25% of vehicle costs and bring an added value of between 40% and 75% [23].

The trend of replacing traditional mechanical systems with modern embedded systems enables the deployment of more advanced control strategies providing additional benefits for the customer and for the environment, but at the same time, the higher degree of integration and criticality of the control application is posing new challenges. These factors are resulting in multiple cross-domain collaborations and interactions in the face of the challenge of mastering the increased complexity involved and also to ensure consistency of the development along the entire product life cycle.

Model-based development supports the description of the system under development in a more structured manner, in the context of handling upcoming issues with modern real-time systems and also in relation to ISO 26262. Model-based development approaches enable different views for different stakeholders, different levels of abstraction and central storage of information. This improves the consistency, correctness, and completeness of the system specification. Nevertheless, such seamless integrations of model-based development still tend to be the exception rather than the rule and often fall short of target due to the lack of integration of conceptual and tooling levels [4]. Consequently, this work focuses on improving the continuity of information interchange from system development level to software development level models.

With this objective in mind the work focuses on improving the continuity of information interchange for architectural designs from system development level (Automotive SPICE [26] ENG.3 respectively ISO 26262 [10] 4-7 System design) to software development level (Automotive SPICE ENG.5 respectively ISO 26262 6-7 SW architectural design). More specifically, the approach is based on the enhancement of a model-driven system engineering framework with software-architecture design capabilities. The model-transformation framework automatically generates software architectures in Matlab/Simulink described via high level control system models in SysML format. The goal is, on the one hand, to support a consistent and traceable refinement from the early concept phase to software implementation. On the other hand, the bidirectional update function of the transformation framework enables facilitation in gaining mutual benefits for the basic software and the application software development from the coexistence of information for them both within the central database.

The document is organized as follows: Section II presents an overview of related approaches as well as model-based development and integrated tool chains. In Section III a description of the proposed bridging approach for the refinement of the model-based system engineering model to software development is provided. An application and evaluation of the approach is presented in Section IV. Finally, this work is concluded in Section V with an overview of the approach.

II. Related Work

Model-based systems and software development as well as tool integration aim at moving the development steps involved closer together and thus improving the consistency of information over the expertise and domain boundaries. Pretschner's roadmap [19] highlights the benefits of such a seamless model-based development tool-chain for automotive software engineering. Model-based development is also claimed to be the best approach to managing the large amount of information and the complexity of the modern embedded systems involved by Broy et al. [4]. Their paper illustrates why seamless solutions have not been achieved so far and mentions concepts and theories for model-based development of embedded software systems. Additionally they make reference to commonly used solutions and problems arising with inadequate tool-chain support (e.g. redundancy, inconsistency and lack of automation). Nevertheless, the challenge of enabling a seamless integration of models into model-chains is still an open issue [20], [21], [27] Often, different specialized models for specific aspects are used at different development stages with varying abstraction levels. Traceability between these different models is commonly established via manual linking due to process and tooling gaps.

The work of Holtmann et al. [9] highlights process and tooling gaps between different modeling aspects of a model-based development process. Giese et al. [8] address issues of correct bi-directional transfer between system design models and software engineering models. The authors propose a model synchronization approach consisting of tool adapters between SysML models and software engineering models in AUTOSAR representation.

Dealing with this gap between system architecture and software architecture, especially while considering component-based approaches such as UML and SysML for system architecture description and AUTOSAR for SW architecture description, is one of the most important topics in this entire issue. Two common variants in the automotive domain are the usage of SysML [3], [8], [11], [14], [17] or X-MAN [12] approaches for architectural description and AUTOSAR for software system description. Boldt [3] proposed the use of a tailored Unified Modeling Language (UML) or System Modeling Language (SysML) profile as the most powerful and extensible way to integrate an AUTOSAR method in company process flows.

The approach of bridging the gap between model-based system engineering and software engineering models based on EAST-ADL2 architecture description language and a complementary AUTOSAR representation is also very common in the automotive software development domain [5], [16], [25]. EAST-ADL represents an architecture description language using AUTOSAR elements to represent the software implementation layer of embedded systems [2]. More recently the MAENAD Project[1] is also focusing this approach.

Kawahara et al. [11] propose an extension of SysML which enables description of continuous time behavior. Their tool integration base on Eclipse and couples SysML and Matlab/Simulink via API.

[1] http://maenad.eu/

Farkas et al. [7] describe in their paper an integrative approach for Embedded Software Design with UML and Simulink. Their presented approach aims in a stepwise migration towards model-based development and enables the co-operative usage of MATLAB/Simulink & UML for functional specification and code generation. The focus of this work is on the combination of source codes generated by different model-based tools, rather than the interchange of data between the different model representations.

SysML and model-based development (MBD) as the backbone for development of complex safety critical systems is also seen as a key success factor by Lovric et. al [13]. The paper evaluates key success factors of MBD in comparison to legacy development processes in the field of safety-critical automotive systems.

Tool support for automotive engineering development is still organized as a patchwork of heterogeneous tools and formalisms [2]. On the one hand, general-purpose modeling languages (such as UML or SysML) provide modeling power suitable for capturing system wide constraints and behavior, but are lacking in synthesizability. On the other hand, special-purpose modeling languages (such as C, Assembler, Matlab, Simulink, ASCET) are optimized for fine granular design, but are less efficient in high-level design.

The issue of improving these interactions, especially those which deal with cross-domains affairs (such as the architectural design refinement from system development level to software development level), thus requires a comprehensive understanding of related processes, methods, and tools. The work of Sechser [24] describes the experiences gained when combining two different process worlds in the automotive domain.

III. Model-Transformation Bridge Approach

This paragraph gives a brief overview of the underlying framework and related preliminary work which supports the proposed approach. The presented framework focuses on improving the continuity of information interchange from system development level to software development level. The basic concept behind this framework is to have a consistent information repository as central source of information, to store all information of all the engineering disciplines involved for embedded automotive system development in a structured way [15].

The methodical support of system architectural design and refinement of this design to software design often fell short of the mark. To handle this situation the AUTOSAR methodology [1] provides standardized and clearly defined interfaces between different software components and development tools and also provides such tools for easing this process of architectural design refinement. Nevertheless, the enormously complex AUTOSAR model requires a high amount of preliminary work and projects with limited resources often struggle to achieve adequate quality within budget (such as time or manpower) using this approach. This approach thus arises out of common AUTOSAR based approaches and forces a direct model transformation from SysML representation to Matlab/Simulink. The reason for making the decision of not fostering an AUTOSAR approach is based on the one hand on

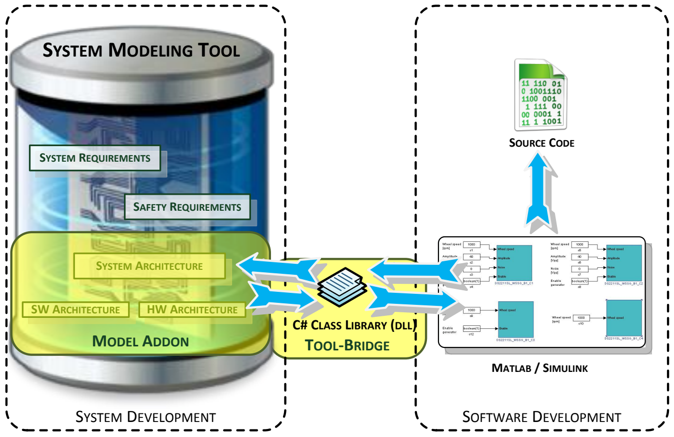


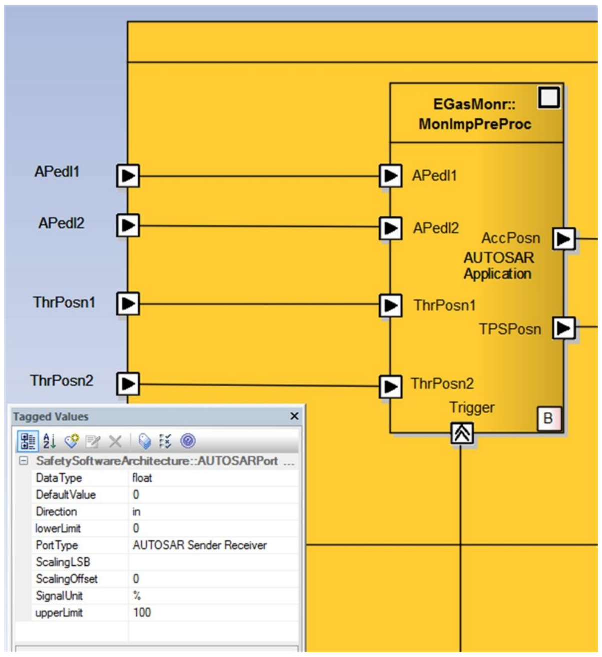
Fig. 1. Portrayal of the Bridging Approach Transferring System Development Artifacts to SW Development Phase

focusing not only AUTOSAR but also rather on generally Matlab/Simulink based automotive software development. On the other hand, experiences we made with our previous approach [14] confirm the problem mentioned by Rodriguez et al. [21]. Not all tools fully support the whole AUTOSAR standard, because of its complexity, which leads to several mutual incompatibilities and interoperability problems. The presented MDB model has been developed using profiles which use a subset of the SysML language to define a SW architecture model particularly tailored to automotive SW engineering in context of ISO 26262. In the following paragraphs we describe the additional model enhancements to support software development and modeling of complex software architectures for function software development. The contribution presented in this work supports automatic generation of software architectures, interface definition, timing setting, and auto-routing of signals in Matlab/Simulink based on SysML representation.

Figure 1 shows an overview of this approach and the imbedded bridging of abstract system development and concrete software development models. More specifically, our contribution consists of the following parts:

- *SW modeling framework*: Enhancement of a SysML profile for the definition of SW component interfaces and SW architecture composition. Required for consistent SW system description, see Figure 1 – model addon.
- *SW architecture exporter*: Exporter to generate the designed SW architecture in Matlab/Simulink for further development of SW functions, see Figure 1 – tool bridge.
- *SW architecture importer*: Importer to integrate refined SW architecture and interfaces from the software development tool (to support round-trip engineering), see Figure 1 – tool bridge.

This proposed approach closes the gap, also mentioned by Giese et al. [8], Holtmann et al. [9], and Sandmann and Seibt [22], between system-level development at abstract UML-like representations and software-level development modeling tools (e.g. Matlab/Simulink or Targetlink). The bridging supports consistency of information transfer between



SafetySoftwareArchitecture::AUTOSARPort ...

DataType	float
DefaultValue	0
Direction	in
lowerLimit	0
PortType	AUTOSAR Sender Receiver
ScalingLSB	
ScalingOffset	0
SignalUnit	%
upperLimit	100

Fig. 2. Screenshot of the SW Architecture Representation within the System Development Tool and Representation of the Interface Information

system engineering tools and software engineering tools and minimizes redundant manual information exchange between these tools. This contributes to simplifying seamless safety argumentation according to ISO 26262 [10] for the system developed. The benefits of this development approach are highly noticeable in terms of re-engineering cycles, tool changes, and reworking of development artifacts with alternating dependencies. As can be seen in Figure 1, the lack of supporting tools for information transfer between system development tools and software development tools can be dispelled by our approach. The implementation of the bridge based on versatile C# class libraries (dll) and Matlab COM Automation Server ensures tool in-dependence of the general-purpose UML modeling tool (such as Enterprise Architect or Artisan Studio) and version in-dependence of Matlab/Simulink through API command implementation. This makes the method especially attractive for projects and companies with limited resources (either in manpower or finances). Small projects or start-up companies in particular often struggle with the problem of setting up their development processes so as to achieve adequate quality.

A. Software Modeling Framework

The first part of the approach is a specific SysML modeling framework which enables the possibility of designing software architectures in an AUTOSAR aligned manner within a system development tool. The profile enables an explicit definition of AUTOSAR components, component interfaces, connections between interfaces and makes the SysML representation more manageable for the needs of the design of an automotive software architecture. Furthermore, it opens up the possibility for defining software architecture and ensures establishment of communication between architecture artifacts with interface specifications (e.g. upper limits, initial values, formulas). Special basic software and hardware abstraction modules are

TABLE I. SW ARCHITECTURE IMPORTER INDICATORS OF TYPE OF CHANGE

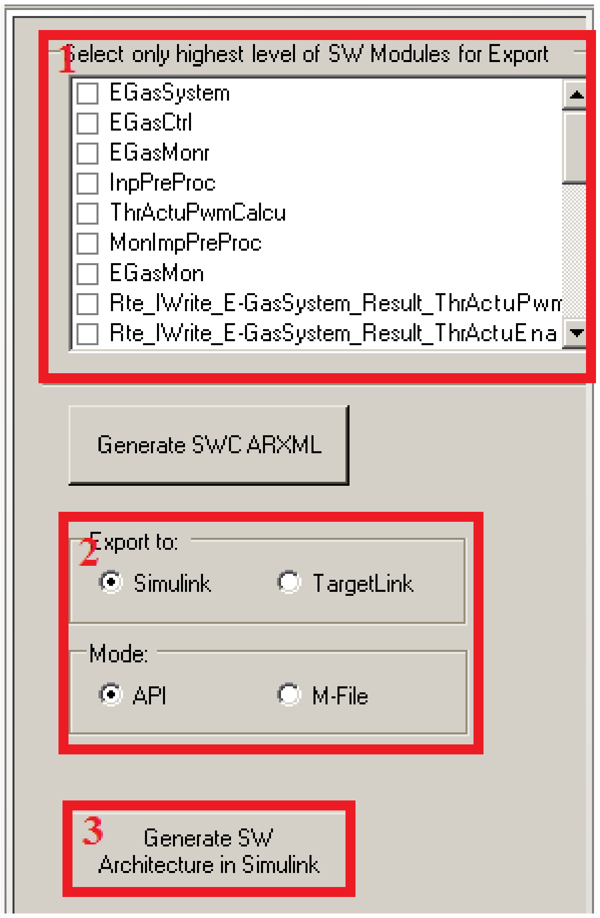
Indicator	Type of Change
A	model artifact added
AC	interface connection added
D	model artifact deleted
DC	interface connection deleted
U	model artifact updated
UC	interface connection updated

assigned to establish links to the underlying basic software and hardware abstraction layers. Moreover, these SW modeling artifacts can be linked to the system model artifacts and requirements in such a manner that traceable links can be established more easily. This has further benefits in terms of constraints checking, reuse, and reporting generation (e.g. for safety case generation). Figure 2 shows an example of software architecture artifacts and interface information represented in Enterprise Architect. Furthermore, this integrated definition of system artifacts and software module in one tool supports the work of safety engineers by adding values and visual labels for safety-relevant software modules.

In addition to standard VFB AUTOSAR profiles the profile features assignment and graphical representation of ASIL to dedicated signals and modules and provides specification of runnables with timing constraints (such as WCET), ASIL, and priority. This additional information enables mapping of tasks to a specific core and establishment of a valid scheduling in a later development phase. Further benefits result in terms of constraints checking and traceability of development decisions.

B. SW Architecture Exporter

The second part of the approach is the SW architecture exporter. The implementation of the exporter is based on Matlab COM Automation Server and generates models through API command implementation, which ensures tool version-independence. The export functionality enables the export of software architecture, component containers, and their interconnections designed in SysML to the software development tool Matlab/Simulink. The SW architecture artifacts to be transferred can be selected by user input and the corresponding Matlab/Simulink model is generated by a background task. As can be seen in Figure 3 the user is able to select the SW artifacts for exporting, the desired model representation in Matlab/Simulink (either TargetLink or Simulink representation), and the exporting mode (m-file based, API based, or as ARXML file). The export mode variants also enable exporting if Matlab/Simulink is not available (m-file based) or an AUTOSAR based SW development toolchain is used (ARXML file based). Listing 1 shows some excerpts of the automatically generated Matlab API commands. As can be seen in this listing, each model artifact, parameter, and connection is transferred to Matlab/Simulink, where the blocks are arranged and sized in a correct manner. Besides this, unique links to the EA representation and assigned safety-criticality marking of the artifact (Listing 1 line 3 and 8) are established.



Select only highest level of SW Modules for Export

- [ ] EGasSystem
- [ ] EGasCtrl
- [ ] EGasMonr
- [ ] InpPreProc
- [ ] ThrActuPwmCalcu
- [ ] MonImpPreProc
- [ ] EGasMon
- [ ] Rte_IWrite_E-GasSystem_Result_ThrActuPwm
- [ ] Rte_IWrite_E-GasSystem_Result_ThrActuEna

Export to: [✓ Simulink] TargetLink

Mode: [✓ API] M-File

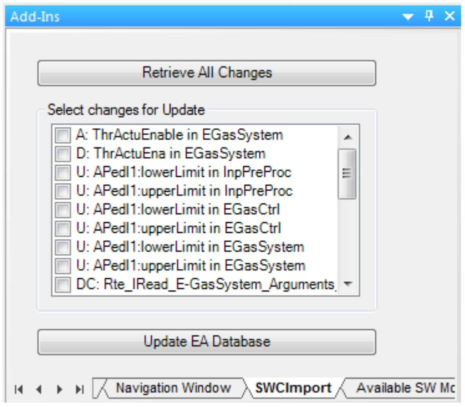
Fig. 3. Screenshot of the SW Architecture Exporter GUI

Listing 1. Excerpts of Matlab API Commands

```
1  addpath(genpath('C:\EGasSystem'))
2  add_block('Simulink/Ports & Subsystems/Model','EGasSystem/
        EGasCtrl')
3  set_param('EGasSystem/EGasCtrl','ModelNameDialog','EGasCtrl'
        ,'Description','EA_ObjectID@1969;ASIL@QM')
4  set_param('EGasSystem/EGasCtrl','Position',[250 50 550 250])
5  :
6  add_block('Simulink/Ports & Subsystems/In1','EGasSystem/
        APedl2')
7  set_param('EGasSystem/APedl2','Position',[50 200 80 215])
8  set_param('EGasSystem/APedl2','Outmin','0','Outmax','5','
        OutDataTypeStr','single','Description','
        EA_ObjectID@1966;ASIL@B');
9  :
10 add_line('EGasSystem','APedl1/1','EGasMonr/1','AUTOROUTING',
        'ON')
11 :
12 save_system('EGasSystem')
13 close_system('EGasSystem')
14 cd ..
15 cd C:\EGasSystem
```

C. SW Architecture Importer

The last part of the approach is the import functionality add-on for the system development tool, which in combination with the export function, enables bidirectional updates of software architecture representations in the system development tool and the software modules in Matlab/Simulink. The



A: ThrActuEnable in EGasSystem
D: ThrActuEna in EGasSystem
U: APedl1:lowerLimit in InpPreProc
U: APedl1:upperLimit in InpPreProc
U: APedl1:lowerLimit in EGasCtrl
U: APedl1:upperLimit in EGasCtrl
U: APedl1:lowerLimit in EGasSystem
U: APedl1:upperLimit in EGasSystem
DC: Rte_IRead_E-GasSystem_Arguments

Fig. 4. SW Architecture Importer User Interface

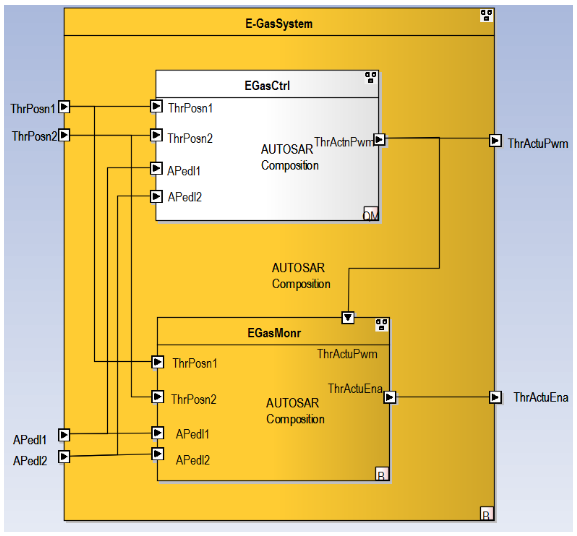


Fig. 5. Top-Level Representation of Demonstration Use-Case in Enterprise Architect

importer analyzes the Matlab/Simulink model representation and identifies the unique links to the EA representation (shown in Listing 1 line 3 and 8). Thereby new and modified model artifacts can be differentiated and changes made in the software development tool can be kept consistent within the system development model representation. This ensures consistency between the models, enables importing of newly available software modules from Matlab/Simulink, and therefore guarantees consistency of information across tool boundaries. Figure 4 shows the user interface within the system development tool. As can be seen in this figure, modifications between the two models are identified and a selective update of the SysML representation can be triggered by the user. Furthermore, a highlighting of the type of change can also be depicted. Table I shows the different change type indicators and types of changes.

IV. Application of the Proposed Approach

This section demonstrates the introduced approach by an automotive embedded system use-case. To provide a comparison and highlighting of the improvements of our approach we use the 3 layer monitoring concept [28] as an evaluation use-case. This elementary use-case is well-known in the automotive domain, but is nevertheless representative. Moreover, this elementary use-case is illustrative material, which is also used for internal training purposes with students and engineers. The disclosed and commercially non-sensitivity use-case is not intended to be exhaustive, nor to be representative of leading-edge technology.

The definition of the software architecture is usually performed by a software system architect within the software development tool (Matlab/Simulink). With our approach this work package is included in the system development tool (depicted in Figure 5). This does not hamper the work of the software system architect, but it enables constraint checking features and helps to improve system maturity in terms of consistency, completeness, and correctness of the development artifacts. Besides this, the change offers a significant benefit for the development of safety-critical software in terms of traceability, replicability of design decisions and it unambiguously visualizes dependencies while putting visual emphasis on view-dependent constraints (such as graphical safety-criticality highlighting of SW modules in Figure 5).

The 3 layer monitoring concept use-case presented consists of 7 SW modules with 34 interfaces and 30 signal connections. Hereby the SW module representations contain 3 configurable attributes per element and the SW interfaces 34 attributes per element. The use-case thus sums up to a total count of 41 model artifacts with 361 configuration parameters and 30 relations between the elements. This elementary example already indicates that the number of model elements and relations between the model elements already becomes confusing. A manual transformation of the information represented within the models would already be cumbersome, error-prone, and would involve a great amount of additional work to ensure consistency between the two models.

The presented approach in this work checks the information and model artifacts for point-to-point consistency of interface configurations before automatically transferring the model representation via 212 lines of auto-generated Matlab API code, which provides evidences and ensures the completeness of the model transformation. The presented SW architecture importer functionality enables round-trip engineering and bi-directional updates of both models and therefore supports evidence for the consistency of both models.

In terms of safety-critical development and reuse the features of the approach presents are crucial to transfer information between separated tools and link supporting safety-relevant information. Moreover, the approach eliminates the need for manual information reworking without adequate tool support, ensuring reproducibility, and traceability argumentation.

V. Conclusion

The challenge with modern embedded automotive systems is to master the increased complexity of these systems and ensure consistency of the development along the entire product life cycle. Automotive standards, such as ISO 26262 safety standard provide a process framework which requires efficient and consistent product development and tool support. Nevertheless, various heterogeneous development tools in use are hampering the efficiency and consistency of information flows.

This work thus focuses on improving the continuity of information interchange of architectural designs from system development level (Automotive SPICE ENG.3 respectively ISO 26262 4-7 System design) to software development level (Automotive SPICE ENG.5 respectively ISO 26262 6-7 SW architectural design). For this purpose, an approach to seamlessly combine model-based development tools on system level (such as Enterprise Architect) and on SW development level (such as Matlab/Simulink) has been proposed.

The applicability of the approach has been demonstrated utilizing an elementary automotive use-case, the 3 layer monitoring concept, which is an illustrative material and does not represent either an exhaustive or a commercially sensitive project. The main benefits of the presented approach are: improved consistency and traceability from the initial design at the system level down to the software implementation, as well as, a reduction of cumbersome and error-prone manual work along the system development path.

ACKNOWLEDGMENTS

This work is partially supported by the EMC^2 and the $MEMCONS$ projects.

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement nr 621429 (project EMC^2) and financial support of the "COMET K2 - Competence Centers for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria, and the Styrian Business Promotion Agency (SFG).

Furthermore, we would like to express our thanks to our supporting project partners, AVL List GmbH, Virtual Vehicle Research Center, and Graz University of Technology.

REFERENCES

[1] AUTOSAR development cooperation. AUTOSAR AUTomotive Open System ARchitecture, 2009.

[2] H. Blom, H. Loenn, F. Hagl, Y. Papadopoulos, M.-O. Reiser, C.-J. Sjoestedt, D. Chen, and R. Kolagari. EAST-ADL - An Architecture Description Language for Automotive Software-intensive Systems. White Paper 2.1.12, 2013.

[3] R. Boldt. Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group, July 2009.

[4] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-based Development: from Isolated Tool to Integrated Model Engineering Environments. *IEEE Magazin*, 2008.

[5] D. Chen, R. Johansson, H. Loenn, Y. Papadopoulos, A. Sandberg, F. Toerner, and M. Toerngren. Modelling Support for Design of Safety-Critical Automotive Embedded Systems. In *SAFECOMP 2008*, pages 72 – 85, 2008.

[6] C. Ebert and C. Jones. Embedded Software: Facts, Figures, and Future. *IEEE Computer Society*, 0018-9162/09:42–52, 2009.

[7] T. Farkas, C. Neumann, and A. Hinnerichs. An Integrative Approach for Embedded Software Design with UML and Simulink. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 516–521, July 2009.

[8] H. Giese, S. Hildebrandt, and S. Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages pp. 555 –579, 2010.

[9] J. Holtmann, J. Meyer, and M. Meyer. A Seamless Model-Based Development Process for Automotive Systems, 2011.

[10] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 1-10, 2011.

[11] R. Kawahara, D. Dotan, T. Sakairi, K. Ono, A. Kirshin, H. Nakamura, S. Hirose, and H. Ishikawa. Verification of embedded system's specification using collaborative simulation of SysML and Simulink models. In *Proceedings of Second International Conference on Model Based Systems Engineering*, pages 21 – 28, March 2009.

[12] K.-K. Lau, P. Tepan, C. Tran, S. Saudrais, and B. Tchakaloff. A Holistic (Component-based) Approach to AUTOSAR Designs. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 203–207, Sept 2013.

[13] T. Lovric, M. Schneider-Scheyer, and S. Sarkic. SysML as Backbone for Engineering and Safety - Practical Experience with TRW Braking ECU. In *SAE Technical Paper*. SAE International, 04 2014.

[14] G. Macher, E. Armengaud, and C. Kreiner. Automated Generation of AUTOSAR Description File for Safety-Critical Software Architectures. In *12. Workshop Automotive Software Engineering (ASE)*, Lecture Notes in Informatics, pages 2145–2156, 2014.

[15] G. Macher, E. Armengaud, and C. Kreiner. Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In *7th European Congress Embedded Real Time Software and Systems Proceedings*, pages 256 –263, 2014.

[16] R. Mader, G. Griessnig, A. Eric, L. Andrea, K. Christian, Q. Bourrouilh, C. Steger, and R. Weiss. A Bridge from System to Software Development for Safety-Critical Automotive Embedded Systems. *38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 75 –79, 2012.

[17] J. Meyer. *Eine durchgaengige modellbasierte Entwicklungsmethodik fuer die automobile Steuergeraeteentwicklung unter Einbeziehung des AUTOSAR Standards*. PhD thesis, Universitaet Paderborn, Fakultaet fuer Elektrotechnik, Informatik und Mathematik, July 2014.

[18] A. Petrissans, S. Krawczyk, L. Veronesi, G. Cattaneo, N. Feeney, and C. Meunier. Design of Future Embedded Systems Toward System of Systems - Trends and Challenges. European Commission, May 2012.

[19] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner. Software Engineering for Automotive Systems: A Roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 55–71, Washington, DC, USA, 2007. IEEE Computer Society.

[20] I. R. Quadri and A. Sadovykh. MADES: A SysML/MARTE high level methodology for real-time and embedded systems, 2011.

[21] E. Rodriguez-Priego, F. Garcia-Izquierdo, and A. Rubio. Modeling Issues: A Survival Guide for a Non-expert Modeler. *Models2010*, 2:361–375, 2010.

[22] G. Sandmann and M. Seibt. AUTOSAR-Compliant Development Workflows: From Architecture to Implementation - Tool Interoperability for Round-Trip Engineering and Verification & Validation. *SAE World Congress & Exhibition 2012*, (SAE 2012-01-0962), 2012.

[23] G. Scuro. Automotive industry: Innovation driven by electronics. http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/, 2012.

[24] B. Sechser. The marriage of two process worlds. *Software Process: Improvement and Practice*, 14(6):349–354, 2009.

[25] C.-J. Sjoestedt, J. Shi, M. Toerngren, D. Servat, D. Chen, V. Ahlsten, and H. Loenn. Mapping Simulink to UML in the Design of Embedded Systems: Investigating Scenarios and Structural and Behavioral Mapping. In *OMER 4 Post Workshop Proceedings*, April 2008.

[26] The SPICE User Group. Automotive SPICE Process Assessment Model. Technical report, 2007.

[27] J. Thyssen, D. Ratiu, W. Schwitzer, E. Harhurin, M. Feilkas, T. U. Muenchen, and E. Thaden. A system for seamless abstraction layers for model-based development of embedded software. In *Software Engineering Workshops*, pages 137–148, 2010.

[28] T. Zurawka and J. Schaeuffele. Method for checking the safety and reliability of a software-based electronic system, January 2007.

Session 22

Multicore & automotive

Thursday 28th, 16:30 – Guillaumet

Optimizing Application Distribution on Multi-Core Systems within AUTOSAR

Wenhao Wang*, Sylvain Cotard*
VALEO Group Electronics Expertise and Development Services
Créteil, France
wenhao.wang@ensea.fr

Fabrice Gravez*, Yael Chambrin*
VALEO Engine and Electrical Systems
Cergy, France
firstname.name@valeo.com*

Benoît Miramond
LEAT Lab, CNRS UMR 7248
University of Nice Sophia antipolis
Nice, France
bmiramond@unice.fr

***Abstract*—Multi-core platforms have gained in popularity in nowadays automotive domain. But, even if multi-core architectures are now supported by the AUTOSAR framework, this migration remains a great challenge. First of all, software designers need new methods to fill the gap between application description and tasks deployment. The use of multiple cores has also to remain compatible with real-time and safety design constraints. Finally, developers need tools to assist them in the new steps of the design process. We propose in this paper a partitioning method integrated in the AUTOSAR design flow acting as a decision guide for the distribution of complex and real world control applications onto automotive multi-core systems.**

Keywords—Multi-core, Partitioning, AUTOSAR, Metaheuristics.

I. Introduction

Nowadays, the multiplication of electronic features in smart engine control implies the execution in real-time of complex computational models. To face this evolution, cars embed ever more ECUs (Electronic Control Units), increasing again the part of embedded software development in the design costs of new generations of vehicles. In the same time, a trend in automotive industries is the adoption of multi-core architecture in critical embedded systems. Now is the time to put it all together by proposing novel design methods facing the scale up of applications, adapting the design process to face the distribution and prediction issues coming from the multi-core advent, while still ensuring the functional safety standards (ISO26262) of the automotive domain.

AUTomotive Open System ARchitecture (AUTOSAR) [1] contributes to meet the increasing complexity in nowadays' automotive electrical and electronic systems. To achieve the technical goals of modularity, scalability, transferability, and function reusability, AUTOSAR standardizes the software development by separating the application and infrastructure. This allows applications to exist and communicate independently of a particular infrastructure. Since its revision 4.0 AUTOSAR has been introducing a new design dimension by supporting multi-core architectures.

On the one hand, regarding the scheduling policies in multi-core systems, AUTOSAR still adopts the static allocation and static priority for the tasks in the system. Static scheduling has been widely studied in the literature, and it remains an efficient way to address the difficult issues of prediction and validation of complex interactions between tasks and shared resources.

On the other hand, multi-core introduces additional challenges that are still difficult to deal with in real world industrial domains where applications exhibit high complexity and special cases features that do not fit with theoretical models. Thus, the shift towards multi-core systems in the automotive industry has revived the challenge of application partitioning to enhance productivity, reusability and predictibility.

This paper proposes a method for the distribution of command and control applications into multi-core architectures, in the purpose of partitioning the computations on the different cores in a near optimal way. We model the problem considering the AUTOSAR specificities and apply metaheuristic algorithms to solve it. This paper presents the first results of such optimization methods on industrial applications of engine control.

The rest of the paper is organized as follows. Section II presents the automotive context and the industrial design flow in which our contribution takes place. We also describe in this section the state of the art on distribution automation in automotive multi-core systems and we explain why current methods are not applicable for concrete automotive projects. Section III presents the formal modeling of the multi-core problem as a Combinatorial Optimization problem. We present our developed tools for partitioning into multi-core platform in section IV. In Section V, we present the automotive case studies considered to evaluate our approach and the quality of the design solutions explored by our tool. Finally, conclusions and future works are discussed in section VI.

II. Automotive Context & Problem Description

A. Automotive application design using AUTOSAR

AUTOSAR mitigates the problems existing in the automotive systems design process by its standardized three-layer software architecture, i.e., the Application layer, the Basic Software layer (BSW) and the RunTime Environment layer (RTE). In the application layer, the applications encapsulate functionalities within a collection of software components. AUTOSAR follows a software component approach as in several description languages. The software components in AUTOSAR (SWC) can interact independently of a particular infrastructure through an abstract environment called Virtual Functional Bus (VFB). Each SWC contains one or more runnables. These runnables are composed of the pieces of codes that can be executed and scheduled independently. Figure 1 depicts such software architecture. All the runnables are triggered by one or several events, such as timing event for periodic runnables [2], data received event for data reading notification and operation invoked event for server (function) calls by clients. The communication between runnables is done by writing and reading the variables. For intra-component communications, these variables are labeled as InterRunnable-Variables (IRV) that can only be shared by the runnables in the same software component. Inter-component communications are realized through Ports and Interfaces.

B. *Configuration of the embedded software*

The implementation of the VFB is realized by the generation of the Run-Time Environment (RTE). RTE is mainly responsible for linking the application to the BSW including Operating System (OS). It also involves the realization of communication between components and the generation of all the RTE events that activate the behavior of runnables.

The configuration of BSW for a specific hardware platform consists in the configuration of the OS and other BSW stacks (communication stacks, memory stacks and I/O stacks). The OS is responsible for the execution of real-time tasks containing executable entities. Each task defines an execution sequence of the runnables mapped to it. The introduction of multi-core in AUTOSAR leads to additional works in the configuration: (1) Software allocation to cores, (2) Task set definition configuration and mapping the runnables to tasks, (3) Variables distribution to memories in the case of hardware architecture with memories hierarchy, (4) Synchronization of the execution flow in multi-core systems.

We consider step (1) and (3) in this paper, as shown in Figure 2. To achieve this goal, real-time scheduling techniques need to be considered in order to adapt the application to the targeted multi-core platform. The considered multi-core platform is composed of a set of 32 bits superscalar cores (from 3 to 8 cores) and a set of associated closely coupled memory local memories. As depicted in Figure 2, data exchanged for inter-core communications are stored at the second level into a shared memory. A typical target is the Aurix architecture from Infineon[1].

In the automotive domain, only static scheduling policies are supported by AUTOSAR. That is, all the runnables are statically attached to cores, and the runnables in each core are scheduled by a local scheduler. One of the core often executes in lockstep the critical parts of the application, which then needs redundancy. Reliability is not considered in the current version of the tool, but will be considered in the future works. In the design flow presented in section III, all these decisions have to be explored by the proposed automation method.

C. *Application partitioning*

In this work, we focus on partitioning applications driven by control and data flow (e.g. engine control, brake control etc.). For that type of command and control applications the order in which the individual statements are executed is very important and the proportion of parallel code is often hard to identify. In consequence, the partitioning of automotive applications into multiple cores requires a fine analysis of the dependencies between runnables and tasks. The paper studies in what extent a design automation method can be employed for that purpose.

D. *Related works*

The theoretical formulation of application partitioning has been widely studied in the past either in the domain of multiprocessor computing [3] or in hardware/software co-design [4]. But the proposed partitioning methods rapidly faced a major limitation considering the lack of real use cases integrated in a full industrial working process. The explored solutions at high-level were too abstract to be really considered. Moreover, when considered alone, the formal optimization clears out the designer from the problem and neglects that not all the design considerations can be theoretically formulated.

In recent years, the adoption of multicore architectures in critical embedded systems has revived the need of design flows fully integrating the exploration phase. So, several works have dealt with the partitioning problem of AUTOSAR applications onto multi-core systems. So, in [5] authors developed heuristic algorithms for mapping runnables into different cores. In this paper, runnables are grouped into clusters before being distributed across cores by optimizing a specific objective function. The works of Faragardi et. al [6] and Saidi et. al [7] proposed a heuristic algorithm to create a task set according to the mapping of runnables on the cores. With the goal

[1]http://www.infineon.com

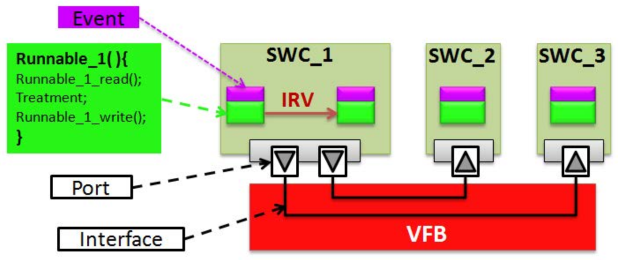


Figure 1. AUTOSAR software architecture

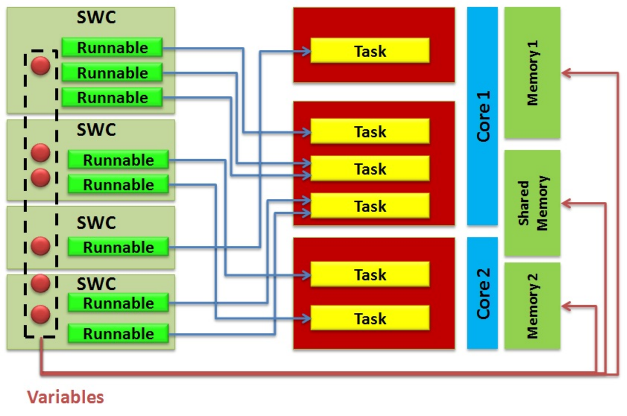


Figure 2. Application partitioning on the targetted multi-core platform

of minimizing the communications between runnables, the problem is classically formulated as an Integer Linear Programming (ILP). Therefore, conventional ILP solvers can be easily applied to derive a solution. In [8], Genetic Algorithms (GA) are applied to partition the application in an optimal way. The results of task allocation are evaluated by their simulation tool TA-Toolsuit. A demonstration version is available in [9] but only simplistic applications are provided. The limitations of the demonstration version avoid any comparison on real applications.
However, all the partitioning methods proposed in the literature only consider the optimization formulation without considering the full design flow. Compared to the existing research work, the proposed method is fully thought into an industrial V-cycle development process. Our contributions are then the following:

- a full working process composed of 5 main phases (see figure 5): application description, dependency analysis, design space exploration, configuration of the executive layer, validation onto the target device;
- back-annotation from the validation phase, enabling optimisation of the cost function from real and credible measurements;
- proposition of a cost function mixing functional and non-functional criteria;
- validation of the solutions explored at high-level thanks to a fully automated refinement process; The detailed description of our working process in section IV will explain how to achieve this goal.

We summarize in table I the properties of the partitioning methods existing in the literature in order to point out our contributions.

Reference	Cost function f	Optimization method	Target architecture	Associated design flow	Validation
[3]	Intercore Communication Overhead (ICO)	ILP	Heterogeneous multicore	No	No
[4]	Total execution time	*SA	Heterogeneous Hw/Sw	No	No
[5]	ICO	heuristic	Automotive multicore	No	No
[9]	Response time, ICO	GA	Automotive multicore	partial	high-level simulation
Ours	Load balancing, ICO, real-time constraints, response time	SA, GA, *TS	Automotive multicore	full	cycle-accurate

Table I. Comparisons of software partitioning methods in the state of the art. *SA is Simulated Annealing algorithm, TS is Tabu Search algorithm

III. Integration of a SW Distribution Method in the AUTOSAR Design Flow

Our proposed automation of the partitioning first asks to formalize the design constraints as a combinatorial optimization problem which mainly relies on the definition of the objective function.

A. *Combinatorial Optimization theories*

Minimizing the objective function involves researching an optimal combination of runnables to cores as well as variables to memories. This problem is assimilated to a Combinatorial Optimization (CO) problem, where solutions are encoded with discrete variables. A model $\mathcal{P} = (S, \Omega, f)$ of a CO problem consists in:

- S: a search space where a finite set of discrete variables are defined;
- Ω: a feasible domain defined by a set of constraints;
- f: an objective function to be minimized.

As the CO problems are NP-hard [10], the complete methods that search for every instance to find optimal solution might need exponential computation time in the worst case. For practical purpose, we often prefer to get a good solution (not the optimal solution) in a significantly reduced amount of time, even though finding optimal solution is not guaranteed. Metaheuristic is this kind of approximate algorithm that aims at exploring the search space efficiently and effectively. This class of algorithms includes - but not restricted to Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithm (GA).

Simulated Annealing is inspired by the physical annealing process of solids. It accepts solutions according to an acceptance probability computed following the Boltzmann distribution $e^{-\frac{f(s')-f(s)}{T}}$, where s' is a neighbor solution of the current solution s, and T represents the temperature.

Tabu search maintains a tabu list and allows adopting the best solution in the neighborhood in condition that it does not exist in the tabu list. This solution is then added into the tabu list after this iteration.

Unlike SA and TS that deal with one single solution at each iteration, Genetic Algorithm treats a population of potential solutions at each iteration. GA uses ideas from biological evolution that includes three main steps: selection, reproduction and replacement. More details on these classical methods can be found in [11], [12], [13].

De-facto, this optimization problem has been modeled in industrial contexts. Reference [14] applies GA to solve the optimization issue of the SWC-to-ECU mapping, and reference [8] applies GA to optimize the task allocation for multi-core processors.

B. *Application and architecture modeling*

The software architecture is modeled using a directed graph $G(V, E)$, such that V is a set of nodes (set of runnables here for AUTOSAR application) and E is a set of edges, also called transitions (links between runnables). A node is modeled as an execution time, a trig mode, a period. A transition has a weight that depends on the size of data transmitted, the period of the producer, etc. The graph size is optimized by the creation of buses between nodes.

We assume that each node V is associated with a period T_i. For the runnables activated by a periodic event, T_i is the period of the activating event. Similarly, for the runnables activated in response to another runnable's result or request, T_i denotes the period of the runnable invoking it or, if it is still not a period event, our partitioning tool identify the one invoking it and so on iteratively.

Each runnable is also associated with execution information that contains two parts: variable accessing time T_a and execution time T_e.

The accessing time T_a mentions the time for a runnable to read or write its related variables located in the memories. In our multi-core architecture, each core is associated with a local distributed memory. Runnables can also access data in shared memories. It is worth to mention that all the memories can be accessed by all the runnables distributed to all the cores, which implies that the accessing time for a runnable to write or read a variable varies with the location of the runnable as well as the location of its variable. In Figure 2, the right part represents a simple model of our architecture with 2 cores: each core has a local memory and there is one shared memory in the system. The accessing time for runnable ρ to access variable θ depends on the location of ρ and θ. All the potential cases are shown in Table II, where $T_{a_\theta}(i,j)$ means the accessing time for ρ located in ith core to access θ located on jth memory. It is obvious that T_a is much shorter if we locate θ into the local memory of the core where ρ is located. Accessing a variable in the local memory of another core is much slower; and accessing to shared memory is dedicated to data exchanged between cores.

The execution time T_e represents the time for a runnable to execute some instructions. T_e is influenced by two factors. One is the performance of the core on which the runnable is located in. The higher computing power, the faster the runnable will finish its corresponding treatment. In a real-life automotive system, the real-time constraints also depend on the execution modes, such as the engine speed or driving modes. E.g. the amount of executed codes depend on the vehicle speed. In the following we denote these contexts cases, and it is the second factor that influences T_e. A weight w is associated to each case to model its importance in the system (high value of w means high importance). So for a given runnable ρ, $T_{e_\rho}(i,n)$ varies with its location (ith core) and the nth case, an example with 3 cases is shown in Table III.

The communications between nodes are presented as transitions E. Each transition contains two nodes ρ_i and ρ_j, $(\rho_j, \rho_j \in V)$, model $\rho_i \mapsto \rho_j$ present the dependency between ρ_i and ρ_j, where ρ_i is the predecessor of ρ_j and ρ_j is the successor of ρ_i. The predecessor ρ_i sends a set of variables that are received by the sucessors. The sum of the size of these variable is noted as S_s. So the sent data rate for the predecessor ρ_i is

$$s_{\rho_i} = \frac{S_s}{T_i} \tag{1}$$

Similarly, the received a set of variable from predecessors. The sum of the size of these variable is noted as S_r, and received data rate for the the successor ρ_j is

$$r_{\rho_j} = \frac{S_r}{T_j} \tag{2}$$

C. Cost function formalization

According to the discussion above, we give the formulation of the problem as follows:

The multi-core architecture is composed of a set of cores $\{\pi_1, ...\pi_I\}$ and a set of memories $\{M_1, ...M_J\}$, with $J > I$ and M_1 to M_I are attached to the local memories of cores π_1 to π_I, while M_{I+1} to M_J represent the shared memories. The partitioning involves the distribution of a set of runnables $\{\rho_1, ...\rho_K\}$ to the cores and also a set of variables $\{\theta_1, ...\theta_L\}$ to the memories. We note $\rho_{k,i}$ when the kth runnable is distributed to ith core and $\theta_{l,j}$ when the lth variable is distributed to jth memory. $T_{a_{\theta_l}}(i,j)$ mentions the accessing time for the runnable located on the ith core to access the variable θ_l located on jth memory. We also define a set of contexts cases $\{K_1, ..., K_N\}$, and w_n is the weight for the nth case. Then, $T_{e_k}(i,n)$ represents the execution time for kth runnable located in the ith core and in the nth case. Thus when we distribute a runnable ρ_k to core π_i, based on its execution time, accessing time and period, this runnable results in a load $u_{\rho_{k,i}}$:

$$u_{\rho_{k,i}} = f\left(T_{a_{\theta_l}}(i,j), T_{e_k}(i,n), T_k\right) \tag{3}$$

The load of core π_i is the sum of the loads caused by the runnables distributed to this core, mentioned as u_{π_i}:

$$u_{\pi_i} = \sum_k u_{\rho_{k,i}} \tag{4}$$

Considering α as the max load ratio of a core, the load of each core must respect

$$\forall i : u_{\pi_i} < \alpha \tag{5}$$

Based on the loads of each runnable in (4) and the weight w_n of each case, we can deduce the load for the entire multi-core system.

The load of the multicore distribution must be well balanced, with a tolerated deviation of 2%. It appears as the main design constraint in the optimisation formulation.

We also define the size of memory M_j as S_j and the size of variable θ_l as S_{θ_l}. The maximum occupation ratio of each memory is noted β. So the occupation ratio of each memory should not exceed it:

$$\forall j : \frac{\sum_l S_{\theta_l}}{S_j} < \beta \tag{6}$$

The intercore communications represent the main challenge to pass from monocore to multicore architectures. They are estimated by summing the number of data access per

Table II. ACCESSING TIME FOR RUNNABLE ρ TO VARIABLE θ

Variable θ	Mem_1	Mem_2	Share Mem
Core_1	$T_{a_\theta}(1,1)$	$T_{a_\theta}(1,2)$	$T_{a_\theta}(1,3)$
Core_2	$T_{a_\theta}(2,1)$	$T_{a_\theta}(2,2)$	$T_{a_\theta}(2,3)$

Table III. EXECUTION TIME FOR RUNNABLE ρ

Runnable ρ	Case_1	Case_2	Case_3
Core_1	$T_{e_\rho}(1,1)$	$T_{e_\rho}(1,2)$	$T_{e_\rho}(1,3)$
Core_2	$T_{e_\rho}(2,1)$	$T_{e_\rho}(2,2)$	$T_{e_\rho}(2,3)$

millisecond. Minimizing this overhead, under load balancing constraints, corresponds to the objective function that evaluates the performance of our partitioning solutions:

$$\mathcal{F} = g\left(u_{\rho_{k,i}}, w_n\right) \quad (7)$$

Equation (7) shows that the cost value of the objective function is decided by the loads generated by the runnable ($u_{\rho_{k,i}}$) in every execution context (weighted by w_n). The loads consider two elements: the CPU utilization computed as $\frac{T_e}{T_k}$ and the communication overhead that is influenced by accessing time of the variables. It is obvious that different ways of partitioning will change the cost value of objective function.

Figure 3 (a) shows a simple example: the application contains 3 runnables ρ_1, ρ_2 and ρ_3. ρ_1 send variable θ_1 to ρ_2 and θ_2 to ρ_3. The hardware model shown in Figure 3 (b) consists in a 2-core system with a shared memory M_3. Besides, each core is attached to a local memory M_1 and M_2. We assume that the execution time for each runnable at each core is identical. The objective is to distribute the application to this 2-core system. Solution in Figure 3 (c) allocates all the runnables in one core, and distributes the variables in its local memory. This could minimize the accessing time, so the communication overhead is low. But the loads of CPU are not well balanced as the other core is empty. Solution in Figure 3 (d) allocates the runnable ρ_3 to the other core, so when runnable ρ_1 finishes its execution, ρ_2 and ρ_3 can execute parallel. Therefore the loads of CPU are better balanced. However, the communication overhead is increased as the accessing time for the variables allocated at the shared memory is much longer. This compromise is considered in our objective function.

In this work, we aim at developing a practical policy for partitioning software applications, composed of several hundreds of nodes, onto multiple cores that will minimize this objective function, while respecting the dependencies and the constraints in AUTOSAR and also verifying the rules in (5) and (6).

D. Description of the optimum solutions searching method

The partitioning solution is represented as a vector in which each element presents the position for runnables or variables. The vector is an ordered list with the length of $l = L + K$, where the L represents the number of the variables and K is the number of runnables to be distributed. In the position i of the vector, $i \in [0, L)$, a memory is distributed for the corresponding variable and in position j, $j \in [L, l)$, a core is attached to the corresponding runnable. The different combinations of the memories and cores will change the value of objective function. In order to deal with this combinatorial optimization problem, we take the metaheuristic algorithms as a solver. The method to search the optimum solution is described as follows:

- the **initial solution** can be obtained in a random way as well as by heuristic guide. The quality of the initial solution would affect final solution;
- the **neighbourhood structure** of a solution defines its possible move direction for improvement, which involves 2 operators: operator $N1$ changes only the memory attached to one single variable to another memory or operator $N2$ changes only the core attached to one single runnable to another core. The move will choose one operator randomly each time;
- the **constraints** guarantee the viability of solutions on each move proposed by the neighbourhood operator: all the solutions (including the initial solution) shall respect all the defined constraints;
- the **metaheuristic algorithms** provide the searching policies to find the optimum (or good) solutions in an efficient way: starting at the initial solution, the improvement is effectuated by a single move (defined by neighbourhood structure) each iteration.

In this work, we apply three metaheuristic algorithms : SA, GA and TS. All the algorithms share the same framework such as initial solution, neighbourhood structure. Each algorithm effectuates different searching policies to find the final solution. The evolution of solutions iteration by iteration is illustrated in Figure 4, which shows the convergence of optimization process by our objective function with the goals that both benefit the acceleration of performance from multi-core and respect the real-time constraints on the dependant tasks.

The results obtained with this method show the contributions of our work :

- the **quality** of the solutions explored according to the cost function;
- the **diversity** of the solutions around the optimum at the convergence of the method. This diversity will

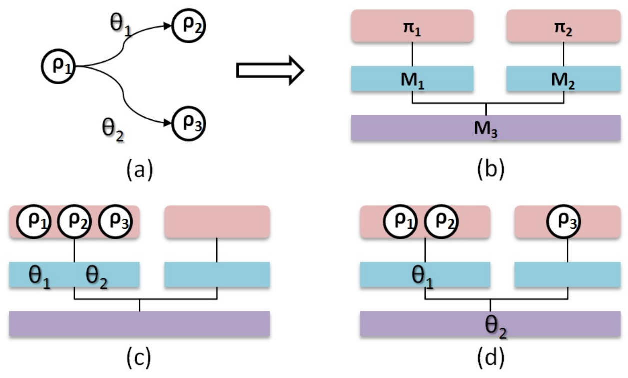


Figure 3. Explanation for objective function (a) Application; (b) Hardware model; (c) and (d) Solutions considering different criteria

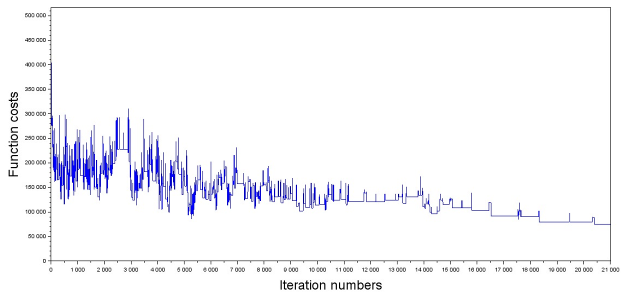


Figure 4. An example of research result by SA

provide the designer the guide needed to take its final decision [15];

- the **scalability** of the method over complex AUTOSAR applications potentially composed of several hundreds of runnables and several thousands of transitions.

IV. Presentation of the partitioning tool

Our partitioning tool presented in Figure 5 is designed to analyze the automotive applications in AUTOSAR and distribute them automatically onto cores. The application targeted in these experiments is composed of a set of software components (SWCs) described in the input AUTOSAR XML files (.arxml). The tool is based on eclipse and written in Java. It allows to analyze a software application by parsing the AUTOSAR XML files. The working process of the tool is described as follows.

A. *Dependency analysis*

As the high sensibility of the execution order and low proportion of parallelism exist in the targeted applications, the partitioning of automotive applications into multiple cores requires a fine analysis of the dependencies between functional elements. For this reason, the tool analyzes the features by the following steps:

- re-works the software architecture: modeling the application as a directed graph presented in the section III-B;
- determines the levels of dependency: building statistics on each transition in the graph;
- analyzes the data information for each transition such as data size, data rate, data unit ;
- identifies the sequences of communications: extraction of data flows.

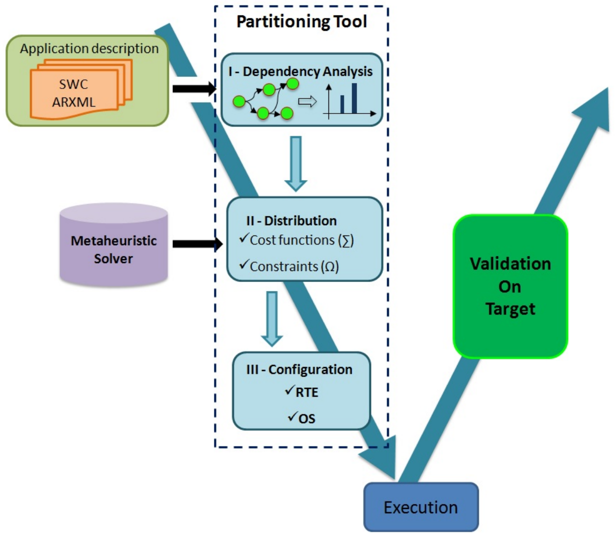


Figure 5. Working process for partitioning automotive application onto multicore architectures

The results of the analysis of dependencies drive the distribution step. More precisely, the level of dependencies and data information are used to evaluate the communication overhead; the sequence of execution would guide the distribution tool to determinate the response time for sequence chains.

B. *Software distribution*

For the distribution part, the tool performs design space exploration (DSE) of the graph designed in dependency analysis step, to distribute the applications into multi-core systems. As stated in the section III, the problem is formalized as a combinatorial optimization problem, which mainly relies on two essential elements: the definition of objective function and a given set of constraints that each solution shall respects. Therefore, applying the metaheuristic algorithms, the tool researches the solutions by evaluating the defined objective function that was presented in (7). Every research step has to respect the constraints presented in (5) and (6).

As to the granularity of element for the distribution, a preparation step is involved in order to minimize the inter connection between the cores. For doing this, the tool determines the dependencies between runnables based on the results obtained by dependency analysis step such as the communication between runnables or the chains of event, etc. Then the tool groups runnables according to the level of dependency between clusters. AUTOSAR SWC is the atomic element that is not allowed to be divided into multiple partitions, thus, all the runnables in the same SWC shall be mapped into the same partition. Respecting this constraint, the tool then gathers again certain clusters into groups. By doing this, we obtain the atomic elements to distribute into cores. These elements are referred as CpuEntities. Then the tool distributes the runnables, or more precisely the CpuEntities, into cores. It also distributes the variables to the different memories. To do this, the tool applies the selected metaheuristic algorithms to find the optimal combination for runnables and cores.

The output of this tool will provide the designer a set of distribution solutions. Each solution is represented as a vector in which each element presents the position for runnables or variables. The designer can then analyze the subset of near-optimal solutions to finally select the best distribution according to non-formalized criteria (designer experience, reusability, management...). For these reasons, we developed our partitioning tool as a decision guide environment. Thus, the expected behaviour of the underlying optimisation heuristics is not to provide only the optimal solution but also the subset of near-optimal solutions.

C. *Configuration of the executive layer*

This step contains the configuration of RTE, OS and other BSW stacks. The partition solutions that provide the allocation information on each core update the configuration of RTE and OS. The configuration of RTE consists in the mapping of runnables into tasks. The configuration of OS includes the terms of priority definition for tasks, tasks partition, allocation of resources, communication and synchronization between tasks. After that, embedded source code of the solution is generated, compiled and downloaded on the target architecture for the final validation of both the real-time and the functional exigencies.

V. Experimental Results

We now describe the experiments leaded to determine the optimization method the best adapted to our context and to validate the explored solutions.

A. Results of dependency analysis

The method has been evaluated with three application descriptions. The first one labeled as App_1 is composed of a small amount of components. This application is built in a random way and the exploration space for this application is exhaustive thanks to its small quantity. Besides, this application contains 3 context cases for the execution time. We have other two applications (labeled as App_2 and App_3) correspond to bigger real industrial use-cases which represent a portion of a full application of engine control. For these two application, we consider only one running execution mode, therefore there is only one context case:

- App_1 contains 15 SWCs with 32 runnables. After analyzing this application, the tool generates 6 CpuEntities with 7 variables;
- App_2 contains 25 SWCs and 208 runnables, the tool generates 14 CpuEntities with about 493 variables;
- App_3 contains 68 SWCs and 562 runnables, the tool generates 21 CpuEntities with about 1358 variables.

The tool also analyzes the transitions information for each application and classifies these transitions according to the different level of dependency. The results for the three tests are shown in Table IV.

B. Results of distribution exploration

The next step consists in distributing the application into a specific multi-core architecture. Our targeted multi-core architecture contains 3 cores, a shared memory and each core is assigned to a local memory. In order to distribute these CpuEntities into 3 cores and the variables into 4 memories, the tool applies the selected metaheuristics: SA, TS and GA. The small application allows us to obtain independently all the possible combinations and to calculate their cost based on (7). Thus we can identify the optimal solution with the smallest cost values among all the potential solutions. The distribution of cost values for all the partitioning solutions of the first application (noted *App_1*) is illustrated in Figure 6. The figure exposes the complexity of the problem even when considering an AUTOSAR application composed of only 32 runnables. The number of feasible solutions exceed several hundreds of thousands solutions (279888 exactly), and so the optimal solution (with value of cost at the left side in Figure 6) only represents $0,0357\%$ of the landscape.

We then apply each algorithm 10 times on application *app_1*. The cost bands of solutions found by each algorithm are compared to the previous distribution of cost values as shown in Figure 6. The more precise results are shown in Table V. GA (in red in Figure 6) always find the optimal solution. SA also find the optimum and other solutions with a cost between $4,02$ and $4,2$. Finally TS never find the optimal solution, but only solutions with costs between $4,1$ and $4,25$. From these results, we can notice that GA can always find the best solution in a longer running time. SA runs faster with a chance less than 50% to find the optimal solution. Considering TS, unfortunately, we never get the optimal solution, but solutions very close to it.

Table IV. Application analysis results

Application	Number of SWC	Number of runnables	Number of transitions	Number of CpuEntities
App_1	15	32	27	6
App_2	25	208	1558	14
App_3	68	562	6826	21

Table V. Optimization results for application APP_1 by GA, SA and TS Metaheuristics. Only GA works on a population size of 10. SA and TS only explore 1 solution per iteration.

Algorithms	Deviation to best solution	Optimal solution finding times/10	Average Run Time (ms)	Number of explored solutions
SA	0.0	4	243.52	1000000x1
GA	0.0	10	279362.09	100000x10
TS	1.97%	0	7467.08	1000000x1

For the two other applications, we considered real industrial use-cases and focus on quantitative results. We applied only SA and GA, as TS does not show its capability to find the optimum for the small application. We remind that we consider constraints of loads balancing for each solution, data for inter-core communication are allocated in the shared memory, and the cost function minimizes inter-core communication overhead (using IOC). With the growth of the application size, it becomes impossible to obtain all the solutions in the exhaustive way as we did on the small application. So, the optimal solution can not be exactly determined. Thus, we used a different criteria to evaluate the quality criteria of the optimization methods. We focused on the standard deviation between the costs of solutions obtained by each algorithm and the cost of the best solution it ever found. The results for the two applications are shown in Table VI and Table VII. From these results, GA can no longer find better solutions than SA. Besides, the run time of GA is much longer. The average run time for both algorithms increases with the size of application, this is shown in Figure 7.

As previously explained, the goal of our partitioning tool is not to still reach the optimum but rather to prune the design space, and only present to the designer the most promising solutions according to a specific objective function. Only the designer can then identify feasible solutions and take the final decision. Nevertheless, from the optimization point of view,

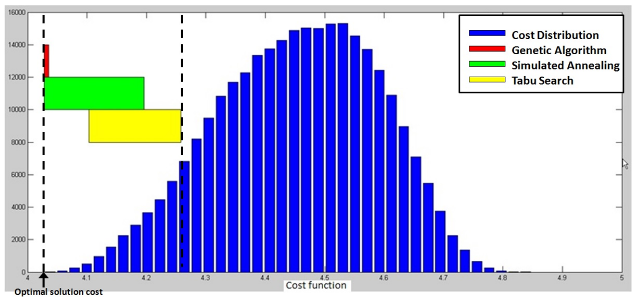


Figure 6. Distribution of the costs of all the partitioning solutions for application *app_1*. The cost band on the left represent the subset of solutions found by the GA, SA and TS methods.

these experiments allowed to identify the algorithm the best adapted to this design problem, even if each of them could be tuned to reach better results. Hence, for this use case, SA shows its ability to provide both the optimal solution and a set of other solutions approaching the optimal one. SA also seems to better scale with the application complexity. The analysis of performances metrics (cores loads, memory occupation, execution time) then allows finer selection.

After the distribution phase, the embedded source code of the solution is generated, compiled and downloaded on the target architecture for the final validation of both the real-time and the functional exigencies.

C. Results of the validation

The target hardware platform is a TC27x tri-core microcontroller. There are two category of memories: the local memories attached to each core and the global memories. There are three cores in this architecture, two identical cores TC1.6P and another core TC1.6E. All these three cores execute the same set of instruction. There are two independent on-chip buses in the tri-core architecture: Shared Resource Interconnect (SRI) and System Peripheral Bus (SPB). The SRI is the crossbar based high speed system bus for TC 1.6.x CPU based devices. The SPB connects the TC1.6 CPUs and the general purpose DMA module to the medium and low bandwidth peripherals. More details can be seen in [16].

We deployed the application *App*_2 onto this multi-core platform to measure the communication overheads and CPU loads for several distributions. After starting the execution, the trace information were obtained by the vendor tool - Lauterbach Trace32. We present in this section the results obtained for two specific solutions:

- initial solution: it is the first generated solution from

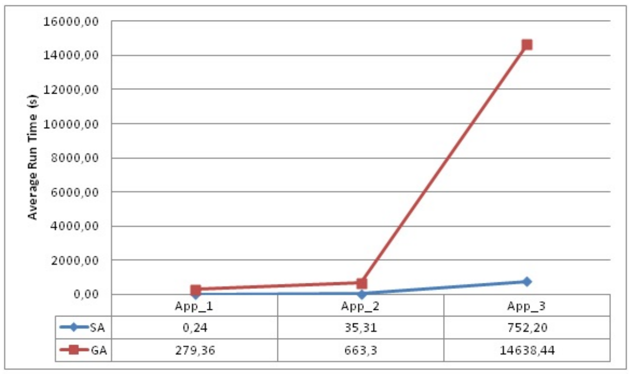


Figure 7. Scalability of the execution time of GA and SA optimization methods. The average run time is plotted according to the application complexity. The figure specifies the average measured values.

Table VI. OPTIMIZATION RESULTS FOR APPLICATION APP_2 BY GA AND SA METAHEURISTICS

Algorithms	Deviation to best found solution	Best solution found	Average Run Time (ms)	Number of explored solutions
SA	0.12%	8	35305	1000000x1
GA	2.83%	7	663305.2	100000x10

Table VII. OPTIMIZATION RESULTS FOR APPLICATION APP_3 BY GA AND SA METAHEURISTICS

Algorithms	Deviation to best found solution	Best solution found	Average Run Time (ms)	Number of explored solutions
SA	21.23%	1	752202.4	1000000x1
GA	10.48%	0	14355693.8	100000x10

which the metaheuristic algorithms search the near-optimal distributions;

- optimised solution: the best solution founded by SA and GA. As shown in the section V-B, the two algorithms could find the same optimised solution for this *App*_2.

The source code of all the solutions found by the exploration tool can be generated and associated to the code of the embedded executive layers. Once compiled, the binary file is downloaded onto the device. We aim at comparing the estimated and real (measured) performances of the explored solutions. The measured communication overhead for the two solutions specifically studied in this paper are given in Table VIII. Estimated values are given by considering the number of data access per millisecond (taking into account the number of fetches required to get data, i.e. the size of data). Measurements are done onto the platform using Trace32 tool and provide the exact amount of time used for intercore communication. It appears in Table VIII as a percentage of the total application execution time. The trace of execution are extracted and analysed in a pseudo-automatic manner. We can for example compute the average load per intercore communication functions (called IOC), and per core by identifying the individual IOC calls, and their execution time, during a period of time.

By comparing real values with estimated values, we can observe that the optimization done by the tool is confirmed by the experiments despite an estimation error. More precisely,

- Table VIII represents the intercore communication cost for each source core (executing the producers of data)
- Table IX shows the associated core loads,

both for the initial and optimised solutions. More precisely, we present in Table VIII the following results of the intercore communications for both solutions:

- the transition counts represent the number of transitions between cores. Each transition is related to 2 IOC functions: send and receive;
- the estimated overhead considers the number of data access per millisecond (taking into account the number of fetches required to get data, i.e. the size of data);
- the measured overhead is the load of IOC functions measured on the target. We can observe in this table that measured overhead is correlated with both transition counts and estimated overhead.

These results show a systematic reduction of the communication and the load metrics, and allow to evaluate the error of estimation.

Table VIII. Estimation and validation results of the communication overhead on the Aurix Tricore target.

	Initial Solution			Optimized Solution		
Cores	transition counts	estimated overhead	measured overhead	transition counts	estimated overhead	measured overhead
Core_0	144	26,25	3,25%	114	26,03	2,0%
Core_1	99	37,20	3,23%	67	22,68	0,94%
Core_2	110	23,50	1,37%	78	15,00	1,2%
Total	353	86,95	7,85%	259	63,71	4,14%
Gain				26,63%	26,73%	47,26%

Firstly, according to the Table VIII, the optimized solutions are better, about 26% more efficient from the partitioning tool point of view, and about 47% in the real platform. It corresponds to about 26% of minimization of the number of intercore transitions. Even if communications are not represented with the same unit in Table VIII we can observe a difference in the global gain. This error of estimation is not very surprising. Performance estimation is currently computed only from the amount of data exchanged between cores. In fact, the count of transitions impacts also the communication overhead. This explains why in Table VIII the decrease of estimated overhead does not necessarily improve the measured overhead while the transition count is increased. Besides, additional features such as the OS services and the memory protection unit (MPU) increase the communication overhead. These overheads should be modeled in the next version of the tool.
Moreover, the on-board profiling showed that, as a system call is done each time the application needs an inter-core communication, it could be more efficient to have 2 data accesses in one communication channel than having 2 communication channels with 1 data access in each. This new optimization will be added as a new type of move (section III-D) during the exploration.

Table IX. Estimation results of the CPU loads on the Aurix Tricore target

Cores	Initial Solution (estimated)	Initial Solution (measured)	Optimised Solution (estimated)	Optimised Solution (measured)
Core_0	4.62%	21,8%	5.34%	20,0%
Core_1	6.51%	21.1%	4.66%	13.3%
Core_2	4.66%	14.4%	5.78%	15.6%
Total	15,79%	57,3%	15,78%	48,9%

Secondly, table IX shows the estimated CPU load for initial and optimized solution. The partitioning tool considers the CPU load balancing as one of the design constraints, and ensures a global load balancing between cores (with a 2% tolerated deviation). The results show that this constraint is respected by the partitioning tool, since based on estimations. The load of cores is measured with Trace32 using dedicated scripts whereas we only consider the load generated by applicative runnables in the estimations. The loads of these runnables were previously measured with Trace 32 onto a single-core distribution (without intercore communication) and back annotated into the application description file.

Thus, the other parts of code executed by the application, such as BSW, OS and other stacks are not considered in the estimations computed by the partitioning tool. On the other hand, real CPU loads are obtained on-board by measuring the time spent in the idle task, and by subtracting the load dedicated to the BSW tasks (main functions). If the current measure provides a best precision compared to high-level estimations, it can still be improved since OS features and other modules are counted in the application load. This explains the differences in the results presented in Table IX. Precisely, we can observe a constant global load according to estimations whereas measures point out the consequences of the distribution onto the core load, due to OS and communication overheads. The execution time of the functional code of the runnables only represents 30% of the global load of this automotive system.

We are now working on adding an intermediate fast validation phase between the distribution and the validation phase to improve the quality of our estimations during exploration. We are developing a SystemC transactional simulator of the multicore software distribution. Besides, similarities between the SystemC language and AUTOSAR have already been demonstrated [17]. At this level, the hardware architecture can be essentially abstracted. The concurrency is modeled at the core level, the goal being to reduce the estimation error on communication costs, to explore more accurately the scheduling of tasks, and to identify in the early phase of the design the conflict of resources. This new simulation step will allow short and long validation cycles in the same multicore design flow.

VI. Conclusion

We described in this paper the issues in the partitioning of engine control applications in multi-core automotive systems. The proposed partitioning method is the first one fully compatible with the constraints imposed by the AUTOSAR architecture both in terms of software architecture and design process. The corresponding partitioning tool can thus be integrated in a seamless AUTOSAR design flow, from application description to software deployment onto multi-core architectures. Hence, classical optimization methods have been adapted to the automotive context and its specific real-time constraints in an efficient exploration tool. The entire working process has been validated onto real world applications from the AUTOSAR descriptions to the on-board profiling.
The results obtained on complex motor control applications show the benefits of the optimization phase. A 47 % gain has been obtained by minimizing the intercore communication. These first results, obtained on the recent intercore release of AUTOSAR, also point out an increase of the core load when migrating from a monocore to a multicore deployment.

After having proposed a pseudo-automatic top-down refinement process in this paper, we aim at recovering the results obtained by real measurements up to the partioning tool in order to improve the precision of the performance estimations. Moreover, thanks to a multi-criteria formulation of the future version of the cost function, we will be able to take into account several criteria to evaluate multicore distributions such as OS overhead, memory usage, resource conflicts, safety...

References

[1] AUTOSAR, http://www.autosar.org, Tech. Rep., last visited: 01/03/2015.

[2] ——, "Specifiaciton of timing extensions, version 2.1.0," www.autosar.org, Tech. Rep., 2014.

[3] Y. Yi, W. Han, X. Zhao, A. T. Erdogan, and T. Arslan, "An ilp formulation for task mapping and scheduling on multi-core architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 33–38.

[4] B. Miramond and J.-M. Delosme, "Design space exploration for dynamically reconfigurable architectures," pp. 366–371, 2005.

[5] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multi-source software on multicore automotive ecus - combining runnable sequencing with task scheduling," *IEEE Trans. Ind. Electron.*, vol. 59, no. 10, pp. 3934–3942, October 2012.

[6] H. R. Faragardi, B. Lisper, and T. Nolte, "Towards a communication-efficient mapping of autosar runnables on multi-cores," *IEEE 18th Conf. on Emerging Technologies & Factory Automation (ETFA)*, vol. 18, pp. 1–5, September 2013.

[7] S. E. Saidi, S. Cotard, K. Chaaban, and K. Marteil, "An ilp approach for mapping autosar runnables on multi-core architectures," *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, 2015.

[8] A. Sailer, S. Schmidhuber, L. Deubzer, M. Alfranseder, M. Mucha, and J. Mottok, "Optimizing the task allocation step for multi-core processors within autosar," *International Conference on Applied Electronics*, 2013.

[9] Amalthea-project, http://amalthea-project.org, Tech. Rep., last visited: 02/02/2015.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman and Company, 1979.

[11] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220 (4598), pp. 533–549, 1983.

[12] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13 (5), 1986.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[14] M. Zhang and Z. Gu, "Optimization issues in mapping autosar components to distributed multithreaded implementations," *22nd International Symposium on Rapid System Prototyping (RSP)*, pp. 23 – 29, 2011.

[15] B. Miramond and J.-M. Delosme, "Decision guide environment for design space exploration," 2005.

[16] 32-bit TriCore Microcontroller, infineon, http://www.infineon.com/, Tech. Rep.

[17] M. Krause, O. Bringmann, A. Hergenhan, G. Tabanoglu, and W. Rosentiel, "Timing simulation of interconnected autosar software-components," *Design, Automation & Test in Europe, 2007.*, vol. 1, no. 6, pp. 16–20, 2007.

Shared SW development in multi-core automotive context

Lothar Michel[1], Torsten Flaemig[2], Denis Claraz[3], Ralph Mader[4]

1 : Audi AG, 84045 Ingolstadt - Germany
2 : Volkswagen AG, Brieffach 39200, D-38037 Braunschweig - Germany
3: Continental Automotive France SAS, 1, av. Paul Ourliac, BP 1149, Toulouse - France
4: Continental Automotive Germany AG, Siemensstr.12, Regensburg - Germany

Abstract

We present a methodology for the common development of combustion engine control Software between TIER-1 supplier and OEM. The classical approach of shared development used in single core projects has to be adapted to the new challenges of integration and protection, in the multi-core context. New integration and protection constraints are specified at design time, which are considered at integration and protection time. A common integration step is defined, where interfaces and constraints at the border are agreed. After that, each part can be modified and protected independently, enabling parallel developments by the partners.

1. Introduction

In the automotive domain, the body controller and chassis systems markets are driven by the integration of new innovative features, resulting in an increase of ECUs in the car, e.g. in an AUDI A8 with up to 80 ECUs. In this context, the multi-core technology is seen as an opportunity to slow-down the inflation of ECUs in the car, by enabling the integration of loosely coupled functions in one same ECU, as a kind of fusion process.

On the other side, the combustion engine market is driven by an increase of engine throughput, a reduction of consumption (CO2), and a reduction of emissions. This results in more complex systems with tighter real time constraints, and finally in SW sizes above 1.5 million of lines of code. Such increase of computation power can only be achieved by the use of multi-core platforms (Fig. 1). The challenge is then in this case to distribute a highly cohesive system on different cores, as a kind of fission process.

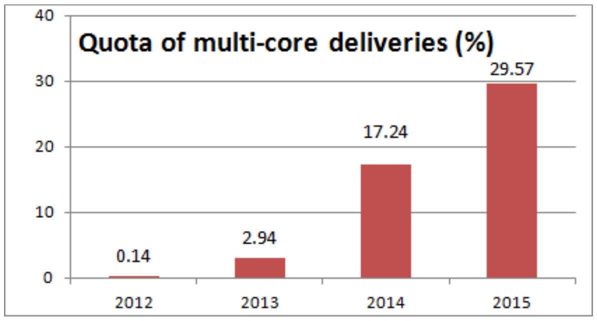


Fig. 1: Quota of deliveries based on multi-core CPU at VW/AUDI

In [1], it has been previously described how this challenge can be handled. We want now to focus on the common work between Continental Automotive, as TIER-1 (supplier), and Volkswagen Automotive Group, as OEM.
In an engine management system, a part of the functions is provided by the OEM, and another part is provided by the TIER-1, resulting in a complex integration of highly coupled SW modules and runnables. In addition, the reduction of time-to-market requires parallel development of these parts, on TIER-1 side and OEM side.

This paper describes the process developed between Continental and VAG to support an integrated shared development in a multi-core legacy (non AUTOSAR) SW, and is based on real project experience. The paper is organized as follows:

In a first chapter, we describe the context of engine control SW. We particularly elaborate on the high coupling of underlying modules, and the hard real time requirements of functions. The iterative development process and the need for parallel development between the parties are also explained.
In a second chapter, the general integration challenge is described. We introduce the concepts developed and in use internally, as well as across partners. We explain that an important step in the integration is the elaboration of a precise and exhaustive cartography of the SW.
In a third chapter, the data protection topic is addressed. We show the importance of this topic, in regard to the high coupling / data flow characteristic to engine systems. The basic mechanisms are developed, from the specification of protection requirements, until the implementation. We finally provide a comparison of 2 basic methods of intervention.
The fourth chapter describes the context of shared development and the different use cases. The need of defining a common architecture, a common integration frame, and the necessary adaptations of the integration and protection processes are explained.
Finally, in a last chapter, we provide the state of the art on these topics, as known from us. We draw a comparison of the standards AMALTHEA, AUTOSAR, and ASAM-MDX, which are the state of the art in the automotive domain. In particular, we point some weaknesses related to the shared development, and to the multi-core aspects, requiring evolutions of the standard.

2. Multi-core challenges at engine systems

<u>Technical context</u>

The importance of integration and data protection in engine systems context is due to the high coupling of combustion control functions, a unique situation in automotive domain. Most of these functions control the same highly dynamic phenomena: the complete thermodynamic process from beginning of air intake till the end of exhaust pipe. The different sensors and actuators interact physically with each other, and therefore the corresponding SW control algorithms permanently exchange information signals.

A rough measurement of this coupling can be based on the number of exchanged signals (SW connectors in AUTOSAR language) (Fig. 2). In most of the cases, the exchange concerns 2 modules. This means one to one coupling (high coupling). In the other side of the spectra, some signals (e.g. engine rotation speed, air temperature…) are needed in many control laws, and therefore exchanged all over the SW. This means 1:n coupling, with n greater than 100 (i.e. 10% of the complete application).

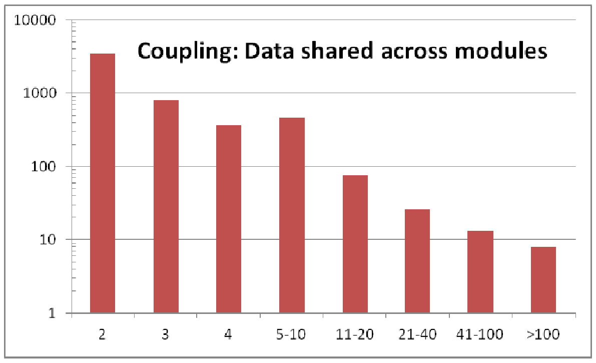


Fig. 2 : Number of modules sharing data

Finally, these signals are data implemented as simple scalars, complex structures, or arrays. For performance reasons, global variables are used.

But this module-to-module coupling gives only an overview of the static facet of the SW. These SW modules are based on several c-functions (executable entities in AUTOSAR language) executed at different rates: for one module, several executable entities might be necessary. Only in 10% of the cases, a single module ends-up in one single executable.

Therefore, ahead to the data flow between modules, a data flow between executables can be measured (Fig. 3), which gives a first idea of race conditions we have to tackle with.

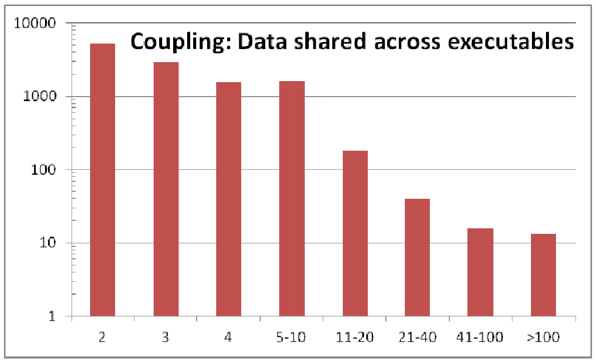


Fig. 3 : Number of executables sharing data

Such estimation gives a similar picture than on component level, but with a higher level of flow: While 70% of the data are encapsulated inside a module, only 30% of them are encapsulated inside one executable.

Finally, the full picture on the data is as follows:

- 1/3 are local to one executable
- 1/3 are exchanged between executables, but local to a module ("inter-runnable variables")
- 1/3 are exchanged between executables of different modules ("sender-receiver")

As these executables (8.000) are scheduled from more than 60 operating system tasks (ranging from 1ms to 1 s, mixing timing and angular frequencies, distributed on different cores), a significant part of the data flow is subject to race conditions in our multi-core environment.

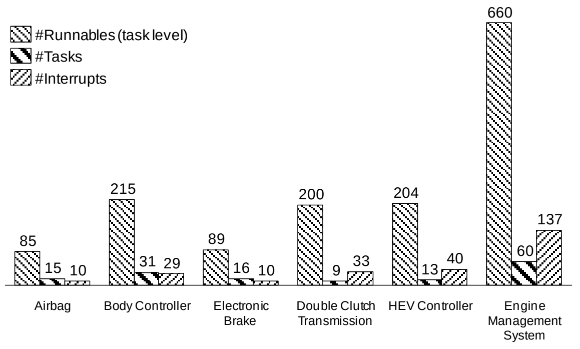


Fig. 4 : Number of dynamic artifacts for integration in different domains

Of course, on the static aspect, the mentioned coupling is reduced using SW composition, to allow a platform/reuse approach. But this has no effect on the race conditions.

<u>Challenges</u>

Therefore, the introduction of multi-core in engine systems is a challenging task.

On the scheduling and integration side, new constraints are adding complexity: The number of integration containers (tasks) increases, the relation between them is more complex (parallelization, chaining...), new types of constraints show up (affinities...), as well as new distribution strategies (all SW on one core, time-dependant SW on one core, safety-related SW on one core...), and the different partners (OEM, TIER-1, 3rd party...) may have different views/constraints on how to utilize the different cores.

On the protection side, a SW running previously in a protected single core cooperative environment (a.k.a. fixed priority with deferred preemption scheduling – FPDS) has now to support parallel execution. In particular, to achieve a maximum flexibility of the SW distribution, which is motivated by the high variability of project configurations, the module designs have to be independent of any core consideration. For instance, 2 runnables of the same module might run on 2 different cores – or not – depending on the project reusing the module. The same module must even still be reusable on the single core projects still under development.

Finally, due to the tight economic constraints, a complete rework of the existing SW is not affordable. Therefore the methodology has to cover the legacy SW not designed for multi-core.

3. Integration process

General

The topic of integration in this context covers the integration of one or more runnables of a software component or composition into one or more tasks of the complete software system, performed by an integrator. In this phase the correct position of the runnable in the sequence shall be determined and the necessary protection of data against concurrent access shall be generated, to ensure the stability or coherency of the data used by the runnable. It is further on assumed that the provider of the runnables and the integrator are time-wise and location-wise separated from each other as a consequence of worldwide software development of large scale software project.

Dynamic requirements for runnables:

For the correct real time behavior of the software functions in the target application, it is necessary to describe unambiguously the dynamic requirements for the integration of the runnables into existing tasks ("dynamic integration"). This is true for runnables of the supplier in context of a platform development and especially for runnables of the OEM, when looking on integration use cases in different applications of the same or even of different suppliers.
Due to this different use cases it is important to describe rather the requirements than a given solution. Describing the solution might be sufficient for one project but would manifest many design constraints to other projects, especially when looking across multiple suppliers using different architectures.

The minimum requirement on dynamic integration is the description of the point in time, when the runnable shall be executed. In the Continental PowerSAR architecture the available points in time ("rates") are standardized in a Reference Architecture and are called SystemEvents. They are characterized by different attributes like required minimum and maximum period, guaranteed deadline, and more. For each Runnable to be integrated, a so-called RunnableEvent is created, which specifies the integration constraint of this runnable by referring to an existing SystemEvent. This RunnableEvent should not be confused with the AUTOSAR RteEvent, which defines more an integration solution than a requirement.
Still missing are requirements concerning the relation between runnables using the same SystemEvent. For the description of requirements for a sequence one could either describe a relation to one or more runnables by name, the so called "Execution Order Constraint" (EOC) or use a requirement concerning the allowed age of a data, which is consumed, named "Data Age Constraint" (DAC) (Fig. 5).

The type of constraint to be used depends on the area of responsibility for a part of the software. Within a software composition where the responsibility is at a single person or small group of developers, EOC might be quite efficient, as giving the order of executables of the own composition combines a lot of refined requirements concerning the data flow and allows an easier description of the sequence requirements. On the other hand most of those execution orders are needed because of the data flow of some "important" data, which contribute to the dynamic behavior of a complete event chain. With EOC this information might get lost and in case of repartitioning or renaming of the software: the EOC might become invalid.

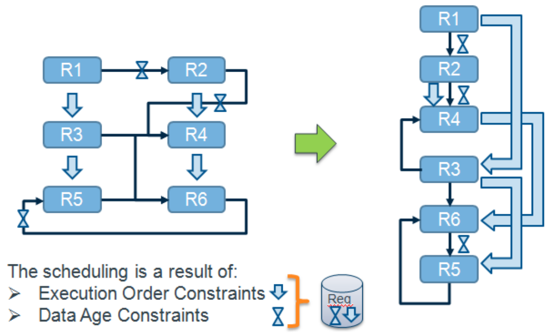


Fig. 5 : Solving Sequence of runnables using Constraints

When using a DAC, one is working on the interface of the runnables and does not depend on runnable names. In a model where only data which are consumed by a runnable are described with a DAC the requirement doesn't become invalid in case of a repartition of the runnables. If the data use is changed, this attribute has to be considered as well. This is especially helpful in a shared development context as the interfaces are the subject of discussion when defining the interaction of OEM and supplier software e.g. in Sequencing Workshops (See Fig. 6).

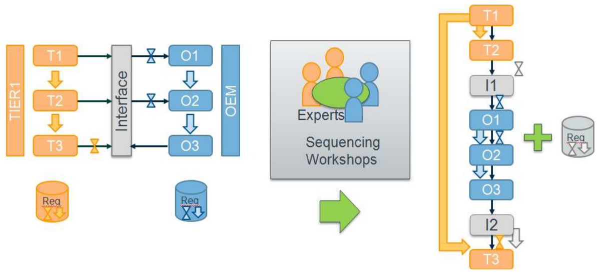


Fig. 6 : Effect of position in the sequence on the data flow between supplier and OEM runnables

Due to the complex coupling and data flow inherent to engine systems, an exhaustive analysis and resolution of data precedence problem is not possible: the problem has not always one solution. Therefore, the principle of the DAC approach is to identify, among all possible flows, the few ones which have a real impact on the system. For instance, low dynamic information, like the air temperature might have one – or more – recurrences of delay without big impact on most of the functionalities. At the opposite, having the wrong value for a cylinder index can be dramatic for the injection controller.

Today this information is exchanged via non formal requirement specification in a textual way due to the fact that MDX V1.2 [2] doesn't offer means for a formal description and most OEMs didn't migrate up to now to AUTOSAR, where the timing extensions would offer a possibility to describe a RTE Event and in addition runnable sequence needs or a DAC.

In addition to these two types of constraints, EOC and DAC, another concept can be used: the phase concept [3]. It consists in partitioning the time domain according to standard features in automatism. As example, logically, the runnables dealing with acquisition and diagnoses need to be executed before the runnables dealing with output commands. Therefore, a phase is associated to a given runnable, according to its inner functionality, and therefore is valid independently of the integration environment. This leads to a fully reusable and independent integration constraint. This method has been introduced at Conti since 3 years now, and used internally. Nevertheless, its extension on shared development with external partners is more difficult.

SW Cartography

For a correct integration and protection of the SW, it is necessary to know the whole structure of the code and the data accesses for read and write performed throughout the whole call graph of the application. No grey area shall be left over, as it can have severe consequences. As precise the cartography will be, as precise the protection and integration will be.
Verifying a DAC is only possible if the full data and control flow of the relevant sequence is known. The same applies to EOC, where the involved runnables are not mandatorily those directly integrated in the sequence.
Depending on the type of the SW which has to be integrated, this cartography can be established based on C-code, MDX description, or ARXML SW-components, a mixture of these use cases needs to be supported.

4. Protection process

Structuring the problem

In terms of data protection against race conditions, two types of problems are taking special importance in multi-core context: data stability and data coherency.

Stability: As soon as runnables can execute in parallel, and exchange data, it becomes very probable, that a data is modified while it is used on another core. If the data is scalar, and atomic, each individual read access cannot be corrupted by a write access on another core. The read access point will get the newest or the old value. But, if there are several read accesses, or if the same value of the data is expected across 2 successive runnables, then the stability of the data might be corrupted. In some cases, this might be acceptable, because for instance, the same data is used in different decoupled parts of the algorithm, and/or because the data has a low dynamics and only a small change of its value is possible. But in other cases, like for instance for booleans or state machines, the impact of a change during an algorithm can be severe.

Coherency: The modification of an elaborated data (structure, array, a set of atomic data…) on one core while it is accessed on the other core can have severe consequences, too. For instance, 2 exclusive information (a flag and its complement…) need to be written and read in a coherent manner. The reading of the data set might be "interrupted" by the writing of these data on this other core. Or on the opposite, the writing of the data-set might be "interrupted" by the reading on the other core. Even both cases can happen. Here again, this does not concern all variables of the SW, but only sub-sets. For instance, in one given algorithm, it is often the case that variables from different rates are used, and therefore cannot be coherent, by essence.

Consistency needs

Considering the huge data flow across runnables and tasks, as depicted in chapter 2, our approach is to be selective on the data and runnables to be protected. Ahead of limiting the HW resources consumption, this limits the protection cases that may have negative functional impacts.

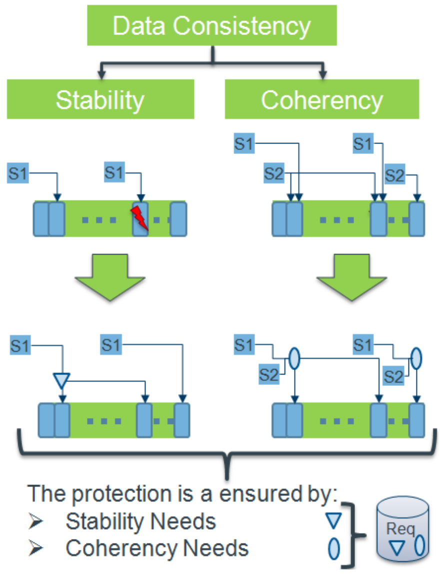


Fig. 7 : Ensuring data protection using Stability and Coherency Needs

Therefore, functional consistency needs are specified at design phase by function experts: stability of certain data accessed by distinct but coupled executables, and coherency for sub-sets of coupled data in certain executable. The function experts are responsible to specify the stability and coherency needs where required, and only there. In this way, the learning and development effort related to multi-core is minimized, and the function experts can concentrate on their core competence: physics and control laws. They have to concentrate on the "what" (to protect) and not on the "how" (to solve the protection).

It has to be noted that the initial AUTOSAR approach ("implicit communication"), where protection was applied everywhere, has been modified in AUTOSAR 4.1.1[4] (and in ASAM-MDX 1.3 [5]), with the introduction of

Data Stability and Data Coherency Needs. But this represents a significant evolution in the RTE (explicit and implicit) communication paradigms.

At integration and protection step, the requirements for data consistency are analyzed and checked against the project architecture (task configuration) and cartography (data flow). In case a consistency is required, a race condition is requested, and the SW is ready to be protected, then the protection of the relevant data is established in the relevant executables.

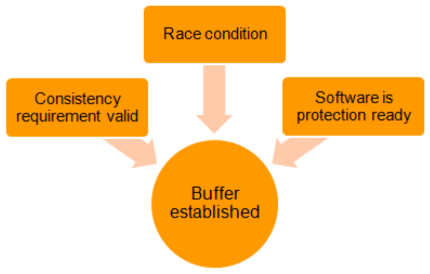


Fig. 8 : Data consistency buffer evaluation

To cover the simple cases where consistency is systematically required, and to reduce the effort of needs specification, complementary automatic strategies are used. For instance, coherency of non atomic (64 bits) data and stability of multiple accesses of same data inside an executable do not need to be specified.

Protection by buffering

The protection of data against race condition is mostly (but not uniquely) ensured by a buffering mechanism. It consists in copying the data in a task-local buffer, and using the buffer instead of the original variable in the algorithm. Fill and flush routines are inserted in the program flow (task bodies) in order to copy the variable into the buffer, and vice versa.
In the AUTOSAR implicit communication, the copies are done at beginning/end of task, resulting in long "buffering segments", and high resource consumption. In classical approaches, the copy is done at beginning/end inside each runnable, which is resource consuming too, but in addition does not ensure consistency across runnables.
In our case, the copies are done along the task to avoid these drawbacks. For instance, the filling of the data into the buffer is done at the latest possible position in the task ("as late as possible"), in order to benefit from the latest available value in the global system.

Access Modification in executable

For modifying the data access to a buffer access in a legacy SW, 2 basic techniques are available:

The Data Reference Modification (DRM) [6] consists in changing the address of the original data by the address of a buffer in the binary ELF file. Here the integration of object code sets some limits in terms of protection. For instance, in case of multi rate executables the function design has to be modified as the DRM process allows data protection in only one context (variable address substituted in binary by buffer address). In this particular case of multi-rate executables, the resulting re-design might have other drawbacks like e.g. code duplication.

	DAM (source code modification)	DRM (binary code modification)
Need of source migration	Yes: Accessers (GET/SET) have to be added to the code. But, a migration can be done by a tool; and code generators can be adapted, in the MBD case. Finally, a similar migration is required for AUTOSAR introduction.	No. But, redesign of the functions might be required in special cases.
Source code exchange	Yes. But some TIER-1 and OEMs are used to work with obfuscated code. In some cases, the OEM functions are even coded by the TIER-1. Finally, AUTOSAR process might need source code exchange, in context of engine systems.	No: object code is the standard for IPP, in legacy context.
Verification and validation	Early: Modification of accessers can easily be checked at compile time.	Late: Need to compile and link before having the modification.
Compiler chain independence	Yes: Accessers modified on source code level.	No: Same configuration of the compiler chain across all partners. Furthermore, the chain has to support DRM technique.
Openness to complex cases	Yes: Step by step, new use cases are supported, which need a more complex redirection of the Accesser, than a simple address modification (e.g. multi-rate cases). Furthermore, the addressing mode to the buffer can be different than the one to the original data, to gain performance.	No: Only address modification can be done, limiting the possible intervention.
Coherency cartography vs. intervention	Yes: the cartography of the SW and the intervention (accesser modification) are based on the same model: The SW-code. This guaranties the global coherency.	No: Unless the cartography is based on the obj. code, which is late in the process, there might be a gap between the cartography and the real implementation in the binary leading to severe mismatches on the protection.

Table 1: Comparison of Data Reference Modification and Data Accesser Modification

The Data Accesser Modification (DAM) consists in modifying the source code. Standardized APIs (data “accessers”) are used in the code, and can be redirected - or not - to the buffer, using an include file. This technique, which is similar to AUTOSAR, gives more flexibility and optimization potential. For instance, it allows runtime context dependant accessers. On the other hand, the different development parties like to protect their IP and are reluctant to share source code, which is necessary for the DAM process.
In Table 1, we provide a non-exhaustive list of pros and cons of each technique.

5. Shared development

Development process

One additional dimension of the SW development for engine systems is the increasing integration, and therefore mixture of SW-components coming from different sources, and often provided with different formats (Fig. 9). It becomes a classical use case, for instance, to integrate SW functions from the OEM in the TIER-1 ECU, as well as components from 3rd party. In extreme cases, the TIER-1 has to integrate SW modules from different OEMs (engine co-developments), or even from own competitors. The amount of external SW may be high (“box-business” model, where the TIER-1 only provides the ECU plus the BSW), or on the opposite, null (“full turn-key” programs). In between these two extreme cases, the SW-part from the OEM to integrate might be provided as models, C-code, or object-code. It might comply with AUTOSAR standard, or simply be legacy SW.

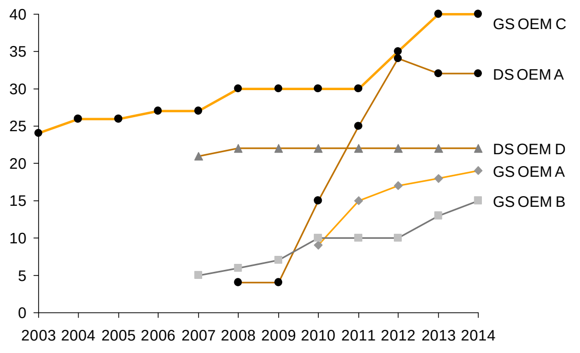


Fig. 9 : Percentage of OEM part in total program code

Of course, this “OEM plug-in” (monolithic or not) has to be integrated in the existing task system, as easily as the rest of the SW. Integration constraints are therefore discussed between the parties. Constraints on the needed SystemEvents are defined, as well as sequencing constraints or event chains. The core distribution is finally derived from all those requirements. The used protection methodology has to give a maximum of flexibility in order to meet all use cases. The OEM plug-in has to be protected against race conditions, like the TIER-1 SW, using the same principles, but with DRM instead of DAM due to IPP reasons.

In total, a complete project lasts in average 24 to 36 months, during which several synchronizations are done between TIER-1 and OEM: releases of the OEM plug-in to the TIER-1; and releases of the complete integrated SW from the TIER-1 to the OEM. Similarly to the TIER-1 functions, the OEM plug-in integrated in the ECU follows a development cycle, and is updated several times during the project life-time. To shorten the development loops, the OEM needs to be able, “at home”, to further develop, re-integrate, and validate his functions. Therefore he must be able to build again the system. This kind of process was a market standard in single core engine systems, where the TIER-1 SW is delivered as object code, and the OEM plug-in is modified, re-integrated, and re-compiled on OEM side without intervention of the supplier. Thanks to cooperative scheduling, a policy extensively used at Continental, there was no need of special mean to ensure data consistency.

Now, as the OEM wants to distribute his SW over different cores, and as the protection of the SW requires a particular analysis and treatment, a segregation of the SW is done, between OEM and Supplier. The TIER-1 part is frozen, protected, and compiled at TIER-1 side, with a first version of the OEM part. Library files are released to the OEM, with a build environment. Starting from this point, the OEM part can be modified, and re-protected at OEM side. This means that independent buffering strategies are applied on the different parts.

At the end, the shared development process can be seen as an alternative or a complement to Rapid Prototyping, enabling short development loops between TIER-1 and OEM.

Common architecture

In this code sharing context, a common understanding of the basic system behaviour is essential to be reached. Right in the beginning, a definition of the features provided by the Operating System has to be negotiated and agreed: a common dynamic architecture. This common architecture has to enable an efficient protection, an easy integration, and it has to support simulation and validation of the scheduling.

First (and already known from single core systems) a set of common SystemEvents, as defined in chapter 3, has to be defined. It might be a subset of the complete set needed by the TIER-1, plus some extensions. Then, these SystemEvents are implemented as tasks, which have a priority and pre-emption behaviour, and a core allocation. In an engine management system typically we have a mixture of time based system events (from 1ms to 1000ms) and angle based system events (crankshaft and camshaft synchronous). Typically, different system events with a same angular period, but different phasing relative to the Top Dead Centre might be required.
But, in addition to periodic system events, sporadic initialization events, such as ECU start-up, ignition key transitions or failure memory clearing have to be specified. Here mainly, the precise position of the event, and its system meaning have to be clear to all parties.

The principle of initialization events is inspired from the object orientation concepts of constructors and destructors, and allows having a coherent system initialization across the complete SW. This concept, used in single core projects, has been enhanced in the scope of multi-core context, as it is important that all cores "toggle" get initialized synchronously and coherently.

Finally, with the step-wise deployment of AUTOSAR, it becomes important to fix also some basis on the use of the RTE, as there are different interpretation/use between partners can cause severe incompatibilities at integration time. Therefore, in the definition of this common architecture, the AUTOSAR configuration has also to be addressed, in particular if the OEM wants to integrate AUTOSAR SW and therefore fixes some implementation choices in its SW-Component descriptions.

For an OEM like VAG the challenge is to find a system setup which is similar in all TIER-1 ECUs, and which is proven to work in a multi-core context. For the above named SystemEvents, this is possible across all ECU suppliers. But the SystemEvents abstract implementation details like OS configuration of priority and pre-emption, core allocation, task chaining and handling of sporadic system transitions (synchronized or not). With multi-core, a new complexity is added to the system.
For a TIER-1 supplier like Continental, the challenge is to find a setup which fits to its generic functions, as far as generic functions are integrated in the project. In effect, in front of the TIER-1, there is a high variability of OEMs, with different visions of the architecture. The TIER-1 challenge is then to fill the gap between the different visions.

Common integration frame

With the ongoing change from single-core to multi-core systems the rising complexity of integration has a huge effect on the software development. It extends the high-level goals of the classic software development, like high reusability, reliability and correctness, IPP. Additional methods are needed, to achieve a closer examination of the sw-architecture. The given heterogeneous tools for integration and protection on the different TIER-1-side must be enabled through standardized general description of integration and protection needs on the OEM side. Additionally it is necessary to consider legacy software, because it is not divisible with further ado. Additional specification and in some cases refactoring is needed.

The main challenge for efficient integration on multi-core systems is an independent partitioning of the software that can run in parallel. But for that it is necessary to find and describe the dependencies and avoid conflicts, to protect and minimize inter-core data-access. Since two years, Volkswagen, Continental and other TIER-1 work together on a model-based approach which resolves the following main question of shared development for multi-core:

- What and how to specify and define the integration and protection needs?
- On which architecture level should the specification be done?
- What is additionally required in methodology and collaborative process?

This model-based approach is a continuous roundtrip from specification until verification [7]. As Fig. 10 illustrates, this roundtrip includes all needed steps in the shared context, considering the typical iterative engineering process in automotive industry, where the software is developed in several iterations of the V-Model and with this has several different releases with different maturity and quality. In a first step to generate the basic element for all considerations in shared context – the system model – a model consolidation combines system descriptions including hardware, operating system and TIER-1-software information given in the AMALTHEA format with the software description including the OEM-software information and requirements in the MDX format. This basis element combines the required information. It is the fundament for shared methods, like the already introduced sequencing workshops. It extends the collaborative software development with new information to be exchanged in new or enhanced formats.

With a consolidated system model, the software architecture can be visualised and graphically annotated with requirements, while analysing the data flow and signal paths. In this second step typically the TIER-1 is responsible to process the requirements and define the final system design. The system-model and defined requirements can be used with tool-support to find a pareto-optimal design, with efficient resource usage and requirement fulfilment. In the third step the updated system design is checked up against the requirements with simulation of the dynamic behaviour in an early design phase, before final integration to take into account, that the design is executable and fulfils all requirements for the given target hardware. Finally in a last step after integration and measurement, the system is verified with evaluation of the requirements taking real software or hardware traces, which includes all system events on required call tree level for referenced elements in the requirements.

In each step enhanced and complex tools are necessary which should interpret the information given from each partner in adequate and standardized exchange formats.

The specification of integration and protection needs, like coherency groups or data ages should be done on an abstract level, considering the design rules of the SW architecture. On SW composition level the requirement engineering is feasible for legacy software and considers the existing development process and given static architecture, because SW compositions already encapsulate functional dependencies given from requirement and architecture engineering. It reduces the costs in development process because complexity and efforts are reduced to specify the integration and protection needs. For that different architecture views have been created, each one fitting best to the use case of requirement engineering.

A static architecture view visualises the logic hierarchical grouping of the static software structure and their interfaces. On this first view it is recommended to define, analyse and check the more static requirements, but also signal grouping for coherency needs are possible.
A second view, the dynamic software architecture view shows more the design of tasks, runnable sequences and their data accesses and data flow. This view is typically used to discuss and analyse dynamic dependencies like execution order constraints, data ages and event chains on runnable level in sequencing workshops.
A new developed view combines both worlds, the static and dynamic architecture: It shows the runnable groups in a SW composition grouped for their period. If it is defined, that this SW-composition specific runnable groups are indivisible (what means the scheduler should not interrupt this group), then it is recommended to define intra execution order constraints for this groups and use data ages for the inter execution order.

Another aspect concerns the compatibility of the SW components to each other. For instance, in order to reach a similar level of parallelism, it would probably be useful to have a similar approach of developing SW components (e.g. rules, patterns) across the parties. Therefore, in a joint research project we look for patterns which are suitable to develop SW components utilizing a high degree of parallelism. These patterns shall be described in a kind of cook book which can be used by OEM developers as well as TIER-1 developers.

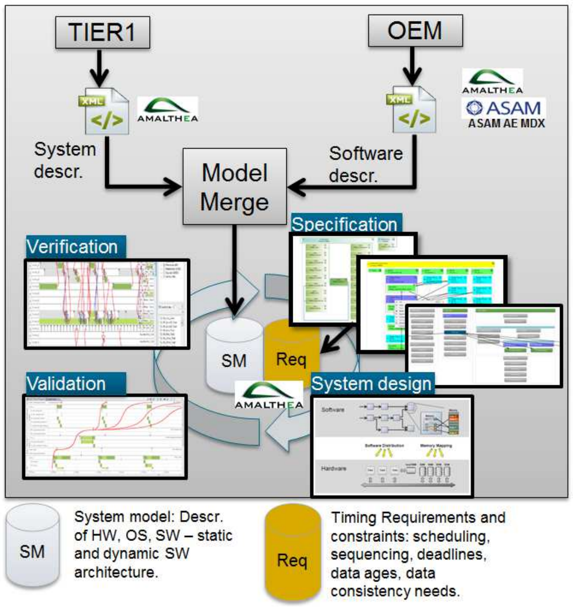

TIER1
OEM
AMALTHEA
ASAM
ASAM AE MDX
System descr.
Model Merge
Software descr.
Specification
Verification
SM
Req
Validation
System design
SM
System model: Descr. of HW, OS, SW – static and dynamic SW architecture.
Req
Timing Requirements and constraints: scheduling, sequencing, deadlines, data ages, data consistency needs.

Fig. 10: Continuous Roundtrip for shared model-based system design in multi-core projects

Then there is the topic of IP protection: even in a close collaboration the different partners need a good level of IP protection. The exchange of source code is maybe not the best solution to reach this goal, unless new technologies like remote build or obfuscated code are used. Object code can be used as exchange format, but with the other drawbacks already mentioned. Due to the permanently increase of inter-penetration of different parties in the final SW, this topic is gaining importance. In addition the exchanged information of the system- and software-descriptions needs IP-protection. For this data it is necessary to obfuscate specified signals and runnable names for IP-parts of the software. But it is recommended to obfuscate only as much as really necessary but not more, otherwise needed information to analyze and specify dependencies and requirements for the interaction of the OEM- and TIER-1 software are lost and in effect the specification in this shared context isn't possible.

<u>Common standard formalism</u>

In order to reach a smooth integration into the defined integration frame the SW description and the specified timing requirements have to be exchanged between the parties. This exchange has to be based on a machine readable standard format, as integration and simulation processes of such complex system are only possible with tool support. Further on the use of a standard helps to define a common understanding of system features and forces the usage of a common wording. As each standard has some space for interpretation, the harmonization of semantics and used tags is necessary.
In cooperation between OEM and TIER-1 the ASAM MDX file format is currently widely used for SW sharing in non-AUTOSAR context. In those projects MDX is used to deliver information necessary for integration purposes to the integrator of a SW component.

The MDX standard is well defined for data definition purposes, as well as for the exchange of SW features information. Also variant handling can be described via system constant definition and settings. Basic scheduling information can be transferred to the integrating party.
As described above, in projects using multi-core CPUs there is the need to exchange further information:

- Data flow information – this currently possible with the existing MDX standard V1.2, but the data access frequency (access multiplicity) has to be added for a more detailed view on the real SW.
- Data stability needs – not defined in V1.2
- Data coherency needs – not defined in V1.2
- Data Age Constraints – not defined in V1.2
- Scheduling requirements for runnables – already possible with V1.2
- Scheduling dependencies between SW components – not defined in V1.2

A new version of the MDX standard has been defined to address the missing topics: the V1.3 released since June 2015.
Data stability groups can be specified as well as data coherency groups using new tags in the SW collection area.
Data Age Constraints and access multiplicity in one executable can be specified on data access elements in SW services (runnables).
With these extensions a SW component provider is able to exchange the defined timing requirements and constraints of its SW components to the integrating party as discussed above.

As there is a lot of legacy code at VW/AUDI this way has been chosen to bring these SW components to the new multi-core world, limiting the effort of reengineering.

After defining this step in exchange format, the focus now changes to:

- implementing of MDX V1.3 in development tools
- gathering all the requirements and constraints to be transported via MDX V1.3
- training the teams to this new process

In the near future the new features of the MDX standard will be used in practice.

The contract = Interface freeze

Freezing of interfaces consists of mainly two steps. The first step happens at the end of interface and sequencing workshops when the interface is agreed between the involved parties and fixed in terms of mapping of content, names, ranges, resolution, DAC and EOC constraints.
The second step is performed, when the interface adaptation is implemented at the TIER1 and the OEM receives software for the parallel development. To manifest and freeze the contract, the software has to be prepared in a certain way.

Protection adaptation: Part Management
When integrating TIER-1 and OEM SW parts, all integration and protection Needs are collected. Each artifact in the project (data, runnable, module…) is allocated to one of the three parts: TIER-1, or OEM part.
This allows applying different buffering policies on the different parts: For instance, if the OEM has not a proper description of the protection Needs (stability, coherency), then an automatic strategy can be applied, which are not necessary on TIER-1 side.
Also, this partitioning allows to select between different formats for the input (C-code, ARXML, MDX), but also for the output (DAM, DRM).

Finally, the final goal of a clear partitioning of the SW is to minimize the interactions between the parts (or at least to concentrate them in an adaptation part) and to enable a partitioning of the protection process. For instance, dedicated buffers, dedicated copy routines, dedicated task sections can be defined and fixed for one part, while the other part is updated. It allows an independent build of the SW at OEM side: The TIER-1 SW is built (protected, compiled, validated) at TIER-1 side, while the OEM SW is re-built as many times as requested, at OEM side.

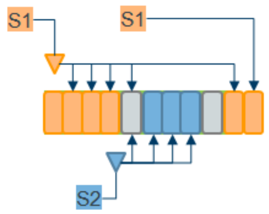


Fig. 11 : Stability Needs apply to different parts

In Fig. 12, we show the resulting buffering for a concrete example. The TIER-1 and OEM runnables are identified by their respective colour. Different buffers are used in the TIER-1 area, which are not reused in the OEM area. The OEM area can then be modified independently of the TIER-1 area, and re-built.

Parallel development / Parallel builds

Having a frozen interface allows starting a parallel development in various stages depending in the amount of changes and the timing requirements of the developed solution. Each partner (OEM, TIER-1) can modify its part without impacting the other one.
When doing only small changes, a parallel development using a tooling for internal bypassing (e.g. eHooks) is appropriate. The limitation of the internal bypass is mainly due to the tight internal resources (RAM, ROM) of the controller.
If the changes grow, or completely new functions are developed, the use of an external bypass system can be useful. With this, an existing function is cut out of the sequencing and replaced by a calculation in an external CPU. The communication to this external ECU is done at defined points before and after the existing function. So the new function gets the same timing environment than the existing one. The communication via separate data buffers to the external CPU ensures stability by default for the external calculated functionality. When using additional variables in the external calculation a coherency need might be not fulfilled.

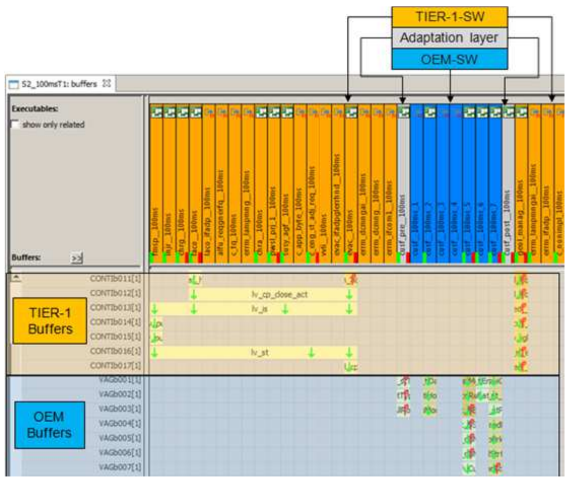


Fig. 12 : Partitionning of runnables and Buffers in shared context

In both cases of external and internal bypassing, if race conditions are modified, the buffering configuration is not anymore valid, due to the modified cartography. It might be that a change in a bypassed function generates a change of buffering in a non modified & stable function. For instance, changing a write access to a read access, or changing the multiplicity of a read access might have impacts on the buffering status elsewhere. Therefore, the type of modifications that can be applied on an algorithm w/o impact on the race condition is very limited, in multi-core context.
In addition, if the required change affects tight integration requirements (e.g. working on a 100µs task in an electrical engine controller), or connection to HW features (e.g. special ASICs), that cannot be fulfilled by rapid prototyping, external and internal bypassing are also no options.

In this case, the solution developed by Continental is the only alternative to the re-delivery of parts between OEM and TIER-1 (with the consequences it has). As mentioned earlier, a high flexibility is provided to the OEM as long as the interface (= integration of the adaptation runnables) is unchanged. It is even possible for the OEM to change its own integration, and in particular investigate different core distributions of his SW.

All the ways to evolve the functionality of the SW system will exist in future. Depending on the need for flexibility and the type of modification, the internal bypass will be a perfect solution for a rapid development. But due to its limitations other solution are needed. On the other hand, the parallel build gives full data protection and guarantees exact real time behaviour, but with the drawback of comparably long turnaround times due to build and flash times.

6. Related works / State of the art

For the discussed topics of shared development of embedded automotive software, state of the art are three quasi industry standards, depending on the use case and target platform in the given project context. The public promoted research project AMALTHEA [8] is of higher interest for software engineering of future multi- and many core software-systems. Parts of the pre-released results with a well-defined data-model from this project are already in use for software architecture specification and description. Currently in usage for exchange more present in the automotive embedded context are the working groups of ASAM MDX [5] and AUTOSAR [4][9]. Unified exchange and interoperability for software description are supported; all data models have equivalent core data.
All these three standards enable software architecture engineering and exchange of relevant information. Depending on the use case, they are more or less suitable. In Table 2, the coverage of the different description context for all discussed use cases is compared for this three standards, where "x" means full support, "(x)" means partly supported and "-" means currently not supported. In this comparison there where considered the latest versions with the highest coverage of information.

About MDX

The MDX description as an ASAM standard is used for single-core projects and with additions since version 1.3 [10]. Also, software description for multi-core projects are supported with the mayor basic multi-core features, like basic timing requirements, data consistency needs and scheduling requirements from the OEM point of view. Complete system description with hardware- and operating system features are not supported. For the use case to describe and exchange the integration needs of the OEM application software it is suitable.

About AUTOSAR

Advanced description and new concepts like Application partitioning and more static architecture support is given with AUTOSAR from version 4.2, which includes the integration- and protection needs and further timing- and architecture description. It is the most established format in the automotive industry, has the most support from architecture and analysis tools and on this reason highly recommended. Nevertheless, the support of consistency needs ("groups") on RTE side is still an open point.

About AMALTHEA

Finally the AMALTHEA format from version 1.1.1 [11][12], as the newest possibility in these cases, adds features and more support for the dynamic description of the software. It extends the architecture and timing requirements and gives possibilities to describe more technical design properties, for example of the target platform with its hardware or the operating system. For use cases like software simulation and partitioning of the software in multi-core context, this is recommendable [12]. This format is suitable for the exchange of complete system description typically generated from the TIER-1 side, which is responsible for the integration.

	ASAM-MDX (v1.3)	AUTOSAR (v4.2)	AMALTHEA (v1.1.1)
Software			
Runnable level			
Data access (i.e. interfaces)	x	x	x
Access occurrences	x	-	x
Runtime	x	x	-
Process level			
Activation (periodic, sporadic, single)	x	x	-
Call sequence of runnables	x	-	x
Hierarchical call sequences	x	-	x
Logic grouping of runnables	x	-	-
Signals			
Data description	(x)	x	x
Requirements			
Execution Order Constraint	x	x	x
Execution Order Constraint (hierarchical)	-	x	x
Data Stability Needs	x	x	x
Data Coherency Needs	x	x	-
Data Age Constraints (time based)	x	x	x
Data Age Constraints (cycle based)	x	x	x
Hardware			
Cores (frequency, instruction per cycle, topology)	x	x	-
Core features (lock-step, peripherals)	x	(x)	-
Memory topology (bus, crossbar, caches, access times)	(x)	(x)	-
Operating System			
Scheduling (algorithm, core resources)	x	(x)	-
Process configuration	x	x	-

Table 2: Comparison of Standards currently in use in Automotive domain

About the automotive domain

The engine systems domain is the first one in automotive requiring an introduction of multi-core processors due to a lack of computing power (with the exception of multimedia). This is the domain where the deployment of multi-core is most advanced, and which has the tightest constraints as mentioned in chapter 2.

About other industrial domains

To our knowledge, there is no other industrial domain where the development of embedded multi-core SW has similar constraints. In aeronautics, space and defense, the time to market, and target system price are not on the same magnitude, like security and safety requirements. As example, the following article shows the growth of automotive embedded SW with the aeronautic case [13]. In particular, in the aeronautic domain, the main focus is on the scheduling topic: due to safety issues, offline scheduling is widely used, which requires a safe estimation of the Worst Case Execution Times (highly impacted by new multi-core architectures). As example, in [14], the authors consider the access to shared resources only in the point of view of timing impact. In Automotive, domain, offline scheduling is not so often used, as the mostly used OS is AUTOSAR OS or OSEK OS. This is even more true in the engine systems domain, where half of the SW is executed at angular (i.e. time variable) rates. Concerning data protection, some studies are conducted, which concern the detection of race conditions, but in our case, we aim not only to identify them, but also to protect them automatically. Furthermore, the shared resource topic is addressed under the view point of impact on timing and WCET. Also, the integration topic is also very specific to engine systems context, as mentioned in the chapter 2 about coupling.

About ARAMIS and ARTEMIS ECSEL EMC^2

ARAMIS:
In [15][16], the authors address the integration topic, but on a vehicle level, and on a basis of distribution of functions across ECUs. Multi-core is seen as an opportunity to reduce the number of ECUs in the car but requires a good distribution of the functions on the cores [17]. In [18], the particularity of engine systems is recognized as it is qualified as a central ECU.
Several papers [19] address the topic of scheduling, of the verification of timing properties. But in general, individual components are considered, designed independently of any framework (reference architecture).
Other papers [20][21][22] address the topic of detecting race conditions, but once again, our purpose is not limited to detection.
ARTEMIS ECSEL EMC^2:
The project EMC^2 is dedicated to multi-core mixed criticality systems, dynamically reconfigurable. The objectives of this project have no relationship to our purpose, as the mix of OEM vs. TIER-1 Sw is not organized on a criticality basis (e.g. TOER-1 Sw low critical and OEM-SW high critical), but rather on a functional basis. Also, the critical/safety aspects are not part of this paper. Concerning the dynamic configuration, and/or reallocation of functions is not seen as a short term option, in regard to the tight coupling and real time requirements in the engine control area. One interesting paper [23] concerns the migration of legacy SW to multi-core platforms, but the approach is different than the one chosen on our side, as a redesign of the functions is requested, according to a pre-established core allocation (which is part of the design itself). In our case, our clear goal is an independence of the design from the integration, which can change from project to project.

7. Conclusion

The formal requirement engineering on dynamic aspects has a relevant impact on the development process and artifacts to be handled. New templates, guidelines and trainings have been set up to cope with these challenges. New design rules are necessary, which will facilitate the parallelization of the control algorithms, but at the same time have to minimize the re-design effort: on OEM side, like on TIER-1 side, the design of the functions have to be prepared for multi-core, but have to be independent of any core and memory distribution, a choice which is highly project specific. It is also not possible to fix core distribution for a function for the next 10 years. To reach this goal of flexibility, it is therefore essential that the function development focuses on the original requirement (protection and integration needs), rather than on any implementation (e.g. using of double buffering).

The introduced model based approach needs long term establishment, but in prototype projects the first experiences confirm significant easement, better system understanding for each party in collaborative process and in conclusion a key enabler to reach the multi-core challenges for SW development. Step by step the process and tools are adapted.

Finally new technologies will arise, which will influence the design of the functions. For instance, dynamic scheduling / allocation to cores, different partitions in the ECU… Also, the increase of computation power linked to multi-core will certainly motivate higher integration of systems, going towards mixed domains ECUs. We can think of course about integration of Transmission Control Unit and Engine Control Unit, PowerTrain Controllers. But it is to be expected that functions out of the PowerTrain domain start to be integrated, leading to an even higher variability, and therefore needs for partial reprogramming, for instance.
At the end, it is doubtless, that the multi-core introduction is at the origin of a big evolution of architectures, and the presented shared development process will be a key enabler.

Works Cited

[1] D. Claraz, F. Grimal, T. Leydier, R. Mader and G. Wirrer, “Introducing Multi-Core at Automotive Engine Systems,” in *ERTS2-2014*, Toulouse, Feb. 2014.

[2] ASAM, *ASAM AE MDX Meta Data Exchange Format standard V1.2.0*, http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1%5Bshow Uid%5D=2559&start=, Dec. 2012.

[3] D. Claraz, S. Kuntz, U. Margull, M. Niemetz and G. Wirrer, “Deterministic Execution Sequence in Component Based Multi-Contributor Powertrain Control Systems,” in *ERTS2-2012*, Toulouse, 2012.

[4] AUTOSAR, *AUTOSAR 4.1.1 Specification of RTE - AUTOSAR_SWS_RTE 3.3.0*, http://www.autosar.org/specifications/release-41/, 2013.

[5] ASAM, *ASAM AE MDX Meta Data Exchange Format standard V1.3.0*, http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1%5BshowUid%5D=3244&start=, Jun. 2015.

[6] R. Bosch, "Method for preventing data inconsistency between accesses of different functions of an application to a global variable in a data processing installation". Germany Patent EP 1 738 257 B1, 30 03 2005.

[7] T. Flämig, H. Jelden, C. Kornmesser, A. Schulze, L. Michel, C. Ebert and B. Cool, *Software architecture methods for multi-core – Distributed development and validation of architecture in collaborative engineered multi-core systems*, EMCC Embedded Multicore Conference Munich 2105, Jun. 2015.

[8] A. Project, *AMALTHEA Project Deliverable: D 4.4 - report on model and tool exchange.*, https://itea3.org/project/workpackage/document/download/1675/09013-AMALTHEA-WP-4-D44-Reportonmodelandtoolexchange.pdf, Apr. 2014.

[9] A. Sailer, *Timing Simulation of Multi-Core Systems based on AUTOSAR Models*, http://www.timing-architects.com/fileadmin/user_upload/Log-In_Miscellaneous/Whitepaper_Timing_Simulation_AUTOSAR.pdf, Aug. 2014.

[10] ASAM, *Release Presentation ASAM AE MDX V1.3.0*, http://www.asam.net/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/documents/standards/AE/MDX/V1.3.0/ASAM-AE-MDX-V1_3_0_Release_Presentation.pdf&t=1446754174&hash=468d38849977145083, Jun. 2015.

[11] AMALTHEA, *AMALTHEA Project Deliverable: D 4.4 - report on model and tool exchange.*, https://itea3.org/project/workpackage/document/download/1675/09013-AMALTHEA-WP-4-D44-Reportonmodelandtoolexchange.pdf, Apr. 2014.

[12] H. Mackamul, *Innovation report: Building an open source, extendible development platform*, https://itea3.org/project/result/download/6729/AMALTHEA%20Innovation%20Report.pdf, Sept. 2014.

[13] R. Charette, "This Car Runs on Code," *IEEE Spectrum*, no. http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code, Feb. 2009.

[14] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet and R. Wilhelm, "Predictability Considerations in the Design of Multi-Core Embedded Systems," in *ERTS-2010*, Toulouse, Feb. 2010.

[15] W. Schwitzer, R. Schneider, D. Reinhardt and G. Hofstetter, "Tackling the Complexity of Timing-relevant Deployment Decisions in Multicore-based Embedded Automotive Software Systems," in *SAE 2013 World Congress & Exhibition*, Apr. 2013.

[16] D. Juergens, D. Reinhardt, R. Schneider, G. Hofstetter, U. Dannebaum and A. Graf, "Implementing Mixed Criticality Software Integration On Multicore - A Cost Model And The Lessons Learned," in *SAE 2015 World Congress and Exhibition*, Apr. 2015.

[17] R. Schneider, A. Kohn and D. Juergens, "Software Parallelization in Automotive Multi-Core Systems," in *SAE 2015 World Congress and Exhibition*, Apr. 2015.

[18] D. Reinhardt and M. Kucera, "Domain Controlled Architecture – A new approach for Large Scale Software Integrated Automotive Systems," in *International Conference on Pervasive and Embedded Computing and Communication Systems 2013*, Feb. 2013.

[19] I. Stierand, P. Rainkemeier and P. Bhaduri, "Virtual Integration of Real-Time Systems Based on Resource Segregation Abstraction," in *International Conference on Formal Modeling and Analysis of Timed Systems, 2014*, Sept. 2014.

[20] D. Nowotka and J. Traub, "Formal Verification of Concurrent Embedded Software," in *International Embedded Systems Symposium, IESS 2013*, Jun. 2013.

[21] T. Ehlers, D. Nowotka and P. Sieweck, "Finding race conditions in real-time code by using formal software verification," in *12th International Conference on Formal Modeling and Analysis of Timed Systems, 2014*, Sept. 2014.

[22] N. Koutsopoulos, M. Northovery and T. Felden, "Advancing Data Race investigation and Classification through Visualization," in *3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, Sept. 2015.

[23] G. Macher, A. Höller, E. Armengaud and C. Kreiner, "Automotive Embedded Software: Migration Challenges to Multi-Core Computing Platforms," in *IEEE INDIN 2015 - International Conference on Industrial Informatics*, Jul. 2015.

Migration of automotive powertrain control strategies to multi-core computing platforms – lessons learnt on smart BMS

Eric Armengaud, Ismar Mustedanagic, Markus Dohr, Can Kurtulus, Marco Novaro, Christoph Gollrad, Georg Macher
AVL List GmbH{eric.armengaud, ismar.mustedanagic, markus.dohr, georg.macher}@avl.com
AVL Research and Engineering {can.kurtulus}@avl.com
AVL Software and Functions GmbH {christoph.gollrad}@avl.com
Ideas and Motion {marco.novaro}@ideasandmotion.com

***Abstract*— An important acceptance criteria for electric mobility is the capability to efficiently use the energy stored in the cells of a battery over the vehicle lifetime. The BMS (Battery Management System) plays a central role by estimating the state of charge (current energy available) and state of health (degradation due to ageing effects) of the cells. Improvement of the estimation quality has a direct impact on the battery and thus vehicle range. It is the target of the INCOBAT project to improve the BMS system by means of new electronic components, new control strategies and new development methods in order to achieve cost reduction and performance (driving range) increase. In this context, the introduction of multi-core computing platforms aim at providing more computing resources and additional interfaces to answer the needs of new automotive control strategies with respect to computing performances and connectivity (e.g., connected vehicle, hybrid powertrains). At the same time, the parallel execution, resulting resources and timing conflicts require a paradigm change for the embedded software. Consequently, efficient migration of legacy software on multi-core platform, while guaranteeing at least the same level of integrity and performance as for single cores, is challenging. In this paper, the lessons learnt during the migration of the BMS control strategies to the INCOBAT BMS computing platform will be presented.**

***Index Terms*—Multi Core, Electric Vehicle, Battery Management Systems**

I. Introduction

IN recent years, electric mobility has been promoted as the clean and cost-efficient alternative to combustion engines. Although there are already solutions on the market, mass take-up has not yet taken place. There are different challenges that hinder this process from an end user point of view such as costs of the vehicle, driving range, or infrastructure support. Several of these challenges are directly connected to the battery, the central element of the full electric vehicle (FEV). The costs of the battery sum up to 40% of the total costs of a FEV, and the driving range of a FEV is strongly reduced in comparison to the combustion engine.

The aim of INCOBAT[1] (INnovative COst efficient management system for next generation high voltage BATteries, started in October 2013) is to provide innovative and cost efficient battery management systems for next generation HV-batteries. To that end, INCOBAT proposes a platform concept in order to achieve cost reduction, reduced complexity, increased reliability as well as flexibility and higher energy efficiency. Moore's law [1], stating the doubling of the computer capacity every 2 years, is still a strong enabler for this fast function increase and at the same time cost-per function decrease. The current development trend for computing platforms has moved from increasing the frequency of single cores to increasing the parallelism (increasing the number of cores on the same die) to limit the power dissipation while improving the performance. Multi-core and many-core technologies have strong potential to further support the different technology domains, but at the same time present new challenges.

Hence, the automotive industry is facing a growing gap between the technologies and required level of expertise to make best use of them. The computing platforms are becoming more and more high-performance with concurrent computing capabilities, larger embedded memories as well as increasing number of integrated peripherals. Low-level mechanisms (e.g., memory protection, diagnostics) typically provided by the basic software or operating system are now being moved into the microcontroller. The complexity of these computing platforms is very high, the related user guides is made of several of thousands of pages. Regarding automotive operating systems and low-level basic software (BSW), the AUTOSAR approach is following a similar trend by standardizing several tens of BSW modules in several tens of thousands pages of specification. Similarly for the application software (ASW, e.g., control strategy for hybrid powertrains), the complexity is already very high and still growing by the introduction of new applications such as advanced driver assistance systems (ADAS) or predictive energy management strategies.

[1] http://www.incobat-project.eu/

Additionally, the functional integration of the control strategies (e.g., transmission with combustion engine and e-drive) further raises the complexity of the resulting application.

The automotive industry is confronted to the central question how to migrate, optimize, and validate a given application (or set of applications) on a given computing platform with a given operating system. A knowledge transfer is required to take over the role of control system integrator and identify the application requirements (both functional and non-functional) and to perform a mapping to the SW and HW architecture. The quality of this mapping has a direct impact on the performance of the control system, and thus of the entire mechatronic system.

Main contribution of this paper is to summarize the lessons learnt during the migration of the existing BMS control strategy to the INCOBAT BMS platform based on multi core technology. The paper is organized as follow: Section 2 introduces the INCOBAT project as well as the BMS platform based on Infineon AURIX^{TM} CPU. Section 3 will present the AVL BMS core functions (state of charge SoC, state of health SoH and State of Function SoF estimation) and the adaptation at the functional level that were performed to take advantage of the multi-core platform as well as to enable migration. Section 4 discusses the migration at software integration level and summarizes performance increase achieved. Finally, Section 5 concludes this work.

II. The INCOBAT project – an overview

The aim of INCOBAT is to provide innovative and cost efficient battery management systems for next generation HV-batteries. To that end, INCOBAT proposes a platform concept in order to achieve cost reduction, reduced complexity, increased reliability as well as flexibility and higher energy efficiency [2].

The targeted outcomes of the project are:

- Very tight control of the cell function leading to an increase of the driving range
- Radical cost reduction of battery management system
- Development of modular concepts for system architecture and partitioning, safety, security, reliability as well as verification and validation, thus enabling efficient integration into different vehicle platforms.

To achieve these ambitious targets, the technical approach chosen in INCOBAT primarily relies on the following 12 technical innovations (TI) regrouped into four innovation groups (see Fig. 1):

- *Customer needs and integration aspects*: ensures a correct identification of customer needs and enables an efficient integration into different platforms. This is supported by the use of mission profiles (TI-01) – in order to take into account the different driving styles of the customers, the different traffic conditions in the same scenarios and the different tracks – and by the integration into a demonstrator vehicle (TI-12)
- *Transversal innovation*: *consistent concept and specification*. This second group targets the optimization of the system architecture and its consistent description over the technologies and over the system hierarchies. This aspect aims at providing a consolidated basis in order to simplify later industrialization of the proposed technologies. This includes the TI-02 "Model-based systems engineering" to improve correctness / completeness / consistency of system specification, the TI-03 "System architecture - efficient partitioning of the functionalities" for system optimization at BMS or even vehicle level and the TI-04 "Integration of multiple functionalities" to reduce the number of electronic control units (and thus related costs) in the vehicle.
- *Technology innovation: E/E control system*: This third group aims at improving the components of the E/E control system. Regarding the electronic parts, it consists of TI-05 "Multicore computing platform for additional computing resources" and the TI-06 "Smart and integrated module management unit". From the software part, this is achieved by the TI-07 "Modular SW platform" and by TI-08 "Improved BMS control algorithms"
- *Transversal innovation: improving system maturity*: This last group targets the evidences related to the trust on the technical solutions with respect to correct operation (TI-10 "Design and validation plan including reliability consideration"), functional safety and security (TI-09 "Definition and integration of safety and security concept") as well as reliability (TI-11 "Reliability and robustness validation"). This group of technical innovations is an indicator for the maturity of the proposed technology and further provides information on the efforts required for proper integration and validation of the system.

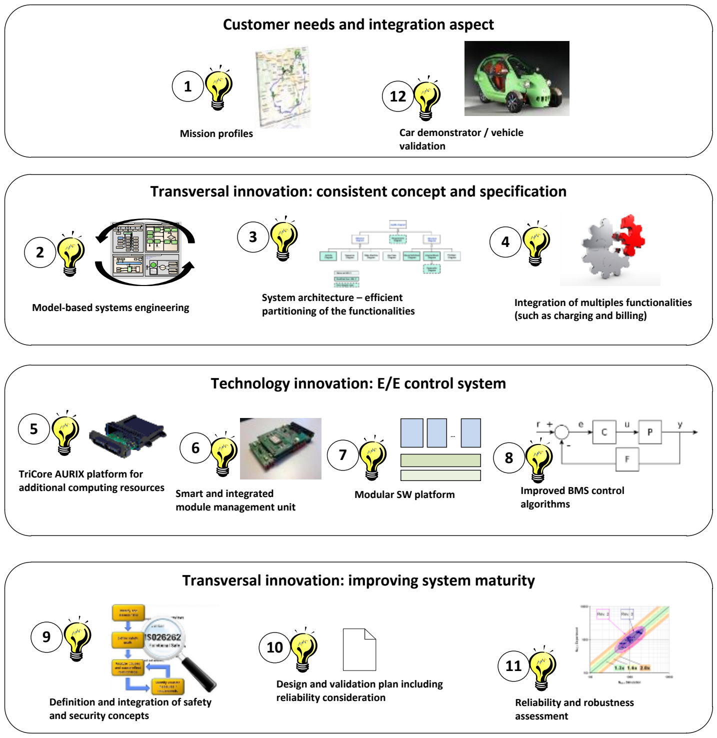

Customer needs and integration aspect
1
Mission profiles
12
Car demonstrator / vehicle validation
Transversal innovation: consistent concept and specification
2
Model-based systems engineering
3
System architecture – efficient partitioning of the functionalities
4
Integration of multiples functionalities (such as charging and billing)
Technology innovation: E/E control system
5
TriCore AURIX platform for additional computing resources
6
Smart and integrated module management unit
7
Modular SW platform
8
Improved BMS control algorithms
Transversal innovation: improving system maturity
9
Definition and integration of safety and security concepts
10
Design and validation plan including reliability consideration
11
Reliability and robustness assessment

Figure 1: Technical innovations within INCOBAT

The INCOBAT BMS CCU (see Figure 2) is based on the Infineon multicore processor AURIX TC275 with an innovative multicore architecture [3]. This device supports the concurrent execution of mixed ASIL functions up to ASIL-D [4]. It offers a rich set of peripherals such as A/D converters and timers for data capturing and it has a reasonable number of IOs to support BMS applications. In conjunction with the specific power-supply ASIC TLF35584 it is possible to supply the CCU and support ISO26262 requirements with a minimum number of components.

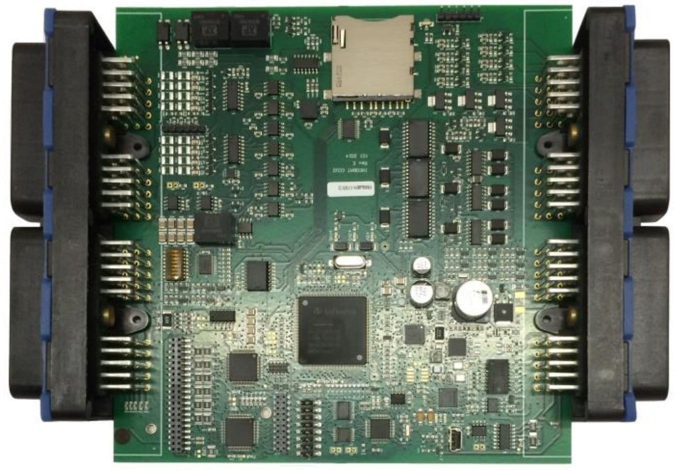


Figure 2: INCOBAT BMS CCU Prototype Hardware

Regarding the SW developments in INCOBAT, a modular development platform is required. Hence, the control strategy and the application software in general are expected to come from different providers and to require different levels of criticality. The activities of SW architect – to define the SW blocks as well as their interfaces – and the activities of SW integrator – integrating the different SW modules and ensuring correct operation of the entire control system – are especially challenging in the context of automotive supply chain with constraints related to functional safety (ISO 26262 [5]). A modular platform is required to enable the distributed development and flexible deployment of different control strategies and applications in an efficient way.

For the SW developments in INCOBAT, the proposed common, modular software development platform consist of:

- ➢ a layered SW architecture, consisting of several layers and components as well as their interfaces, providing access for the applications to the underlying HW capabilities,
- ➢ a suitable SW development tool chain, which supports the application SW developers by means of an effective and consistent development process to seamlessly integrate their particular applications to an overall BMS.

III. Functional migration of the BMS control strategy

A. *Model-based battery state estimation*

1) Introduction

Accurate estimation of battery parameters such as SoC, SoF and SoH requires model-based estimation methods in which a representative model of the battery is utilized as part of the algorithm. Incorporation of a battery model enables many possibilities including:

- Capturing expected behavior, to be compared with actual measurements for inference of parameters causing the deviation,
- Ability of algorithms to handle different operating conditions and usage scenarios,
- Generation of a "prediction error" signal that is necessary for modern estimation methods such as Kalman Filters, or other similar types of Observers.
- Ability of the algorithms to be adapted to different cell types, and cell chemistries via only adapting the incorporated battery model
- Possibility of making predictions of various battery behaviors as well as battery condition

Model based approach to battery state and parameter estimation, therefore has many advantages.

An overview of how a battery model may be used as part of the algorithms is given in Figure 3. This architecture uses the output of the battery model for the prediction step of the estimation method, and compares it with the actual measurement to generate an "error" signal. This error signal is then utilized as part of the algorithm to calculate important signals such as SoC and eventually to update the model and adapt it to the actual battery and operation conditions. This architecture is what enables adaptation to various operating conditions and handles situations where the model may not represent battery behavior accurately.

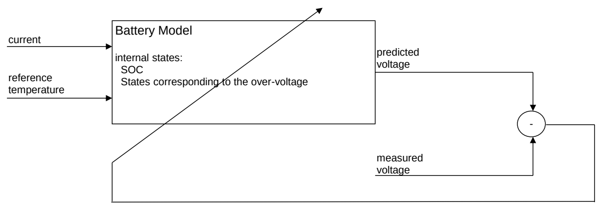


Figure 3: Overview of Model Based Estimation for Battery States

The battery model has to be constructed and parameterized in accordance with the type of cell used in the target application since there are large differences of behavior between various types. The model parameterization requires test data that captures the behavior of the cell (e.g. terminal voltage response, surface temperature) under various operation conditions in terms of ambient temperature and usage (i.e. load current).

However, there are certain drawbacks included with usage of model based estimation techniques, such as relatively high computation power demand, necessity of a high quality parameterization to obtain satisfactory performance and the necessity to ensure stable behavior over all possible operating conditions.

2) *SoC Estimation*

The State-of-Charge is defined as the percentage of the maximum possible charge that is present in the battery. The SoC can't be measured directly, but an accurate SoC on pack and on cell level is mandatory for the energy management control system (State of Function Calculation). Several methods have been developed in the past:

- Coulomb counting (Ampere-Hour Counting)
- Open circuit voltage
- Neural Networks
- Heuristic interpretation of measurements, Fuzzy Logic
- Model-based estimation methods
    - Kalman Filter for battery SoC determination
    - Luenberger Observer
- Sliding-Mode Observer
- LPV Observer

A detailed description of available methods for SoC estimation is given in [6]. Furthermore, common algorithms for State-of-Health (SoH), State-of-Function (SoF), Remaining useful Life (RUL) are introduced.

Nowadays, the most widely used SoC-estimation algorithms are Kalman-filtering techniques. They are based on an equivalent circuit model (ECM) of the battery.

The ordinary KF can be used for linear models. Since the battery is a highly nonlinear system, a Kalman filter is necessary, which permits the use of nonlinear models. The Extended Kalman Filter has the ability to handle such kinds of models. A big advantage of a KF is that it considers measurement and process noise due to voltage sensor inaccuracy, temperature fluctuations etc. to estimate the states of the battery.

One of the first use of an EKF (Extended Kalman Filter) for SoC estimation of lithium-batteries is published in [7], [8] and [9]. These papers describe the mathematical background, the modeling of the battery with its identification requirements and the final implementation of the EKF for SoC estimation. Additionally, an algorithm for SoH estimation is presented.

The publications [10] and [11] describes a modified KF. The Sigma-Point Kalman filter is a more accurate estimation approach, although the computational demand is of the same order as EKF. A comparison between SPKF and EKF show an improvement of the estimation. Moreover, an advanced algorithm is presented, where state and parameter estimation is done simultaneously. A summary of the mentioned algorithms is presented in [12].

A slightly modified Kalman filter with lower computational demand is used for SoC estimation as part of INCOBAT. The algorithm runs on module level.

The SOC-function estimates:

- Module States: States of the battery model (model has 6 states, including SoC of the battery)
- Module State of Charge
- Module OCV: Open circuit voltage of the module

The relatively high computational demand makes it very challenging to run the KF on cell-level with state-of-the-art battery management systems (BMS), especially for battery packs intended for high voltage applications. That is why SOC-estimation is done on module level.

3) *SoH Estimation*

The State-of-Health is a measure of the condition of the battery compared to the fresh battery. It is characterized in the loss in capacity and the increase in resistance.

To assure correct SoF and SoH estimation of a battery pack, it is necessary to have information of the cell-SoC and the cell-resistance as well. An algorithm to estimate these values is described in [13]. The basic mathematical background of the estimation approach is described in the next sections.

The cell observer is used to determine the deviation of each cell compared to the mean module state already estimated as part of the SoC function and the module resistance observer. The aim is to utilize a much simpler linear algorithm for computational efficiency.

The used model is a linearized model that describes the deviation of each cell from the module mean rather than full cell dynamics. The algorithm is a modified version of the Recursive Least Squares, a well-established algorithm commonly used in state and parameter estimation problems.

4) *SoF Estimation*

The State-of-Function consists of measures for the ability to fulfill the application specific function of the battery. The BMS has to calculate current and voltage limitations such that the battery is operated in a safe operating mode and the performance and lifetime targets are met.

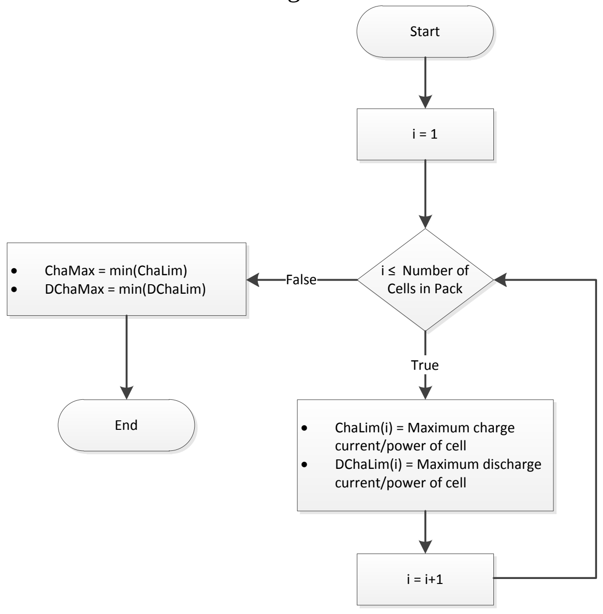

Start
i = 1
i ≤ Number of Cells in Pack
False
True
ChaMax = min(ChaLim)
DChaMax = min(DChaLim)
End
ChaLim(i) = Maximum charge current/power of cell
DChaLim(i) = Maximum discharge current/power of cell
i = i+1

Figure 4: State of Function Flow Chart

State of Function calculation is responsible for determining available functionality of the battery, which would be either current or power that can be supplied to the powertrain, considering the maximum allowable cell voltage and the maximum allowable operating current. The SoF calculation is based on a prediction of future cell voltages for a calibrated prediction time with an electrical model.

Basically, the limits are calculated for each cell, as it is shown in Figure 4. The flowchart shows an iteration over the maximum number of cells in the battery pack. For SoF limits,

the results of the 'worst' cell (e.g. highest inner resistance) are taken into account. In more detail, the calculation estimates a current limit, based on the average cell voltage in a module. Afterwards, the limits are corrected by the worst cell in a module. The worst cell is identified by the calculated dSoC and dR values.
The algorithm provides the maximum charge and discharge current limits, as well as the maximum charge and discharge power. The power limitations can be calculated by a multiplication of the current and voltage limits.

5) Function inter-dependencies

Figure 5 provides an overview how the algorithms for SoC, SoH and SoF estimations are implemented. SoC and SoH estimations run on module-level due to the high computational demand, whereas the SOF are calculated per cell. The battery pack SoC is simply the average of the estimated Cell SoC values. To get a plausible SoH of the module resistances of the battery pack, delta SoC and delta R should be considered.

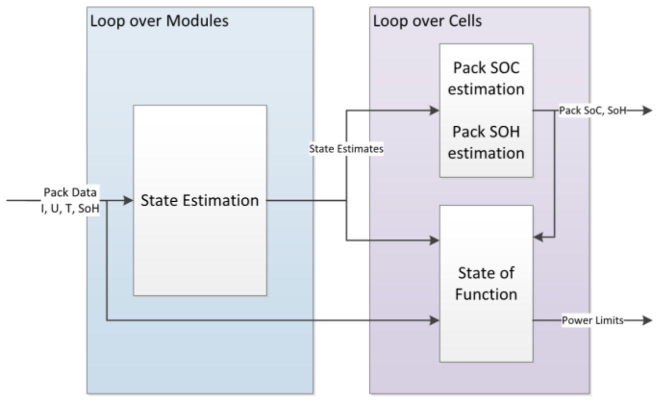


Figure 5: Information flow of estimation algorithms

6) Functional migration to parallel computing scheme

From the point of view of the model-based battery state estimation, the main focus is on the analysis of the existing algorithms (currently running on single core computing platform) in order to identify possible improvements with respect to SoX estimation accuracy while making use of the additional computing power of a multicore processor. The main target is to improve the accuracy of the estimations from a group of cells (e.g. a module) down to single cell level. In our case, the sensor platform already provides the information required to measure the cells. The challenging factor here is to run a dedicated instance of the existing model estimation algorithm for each single cell instead of one instance for a group of cells. The required computing power (number of algorithm instances running in parallel) is therefore directly dependent on the modelling accuracy (reduction of the number of cells taken into consideration for one algorithm instance). A higher modelling accuracy provide more accurate information on the status of the cells, thus moving the limits for the use of each single cell (and therefore of the entire battery at the end) from a conservative boundary to a more real limit. Consequently, it can be assumed that the range of the vehicle and cycle life of the battery can be increased due to the precise estimation approach.

B. Electrochemical Impedance Spectroscopy (EIS)

1) Introduction to EIS

This methodology measures dielectric properties of a medium as a function of frequency. In particular, as applied to the battery cells, the goal of the EIS is to determine the impedance parameters, and the state of health (SoH) of the cells as a function of the impedance. In order to successfully determine the EIS spectrum it is necessary to take into account certain inherent problems in the method and the component under test.
The EIS analysis is based on the following prerequisites:
a) the system must be linear
b) the system parameters should not vary over time
c) the system must be single input, single output (SISO)
The lithium battery is not generally satisfying these requirements: therefore, additional assumptions have to be made.
First, the characteristic of a battery is not linear: To calculate the impedance it is therefore necessary to proceed to a linearization. The used technique is to identify a working point on the electrical characteristic and to generate a small perturbation of it. Analyzing a small enough portion of a cell's current versus the voltage curve, it is considered to be linear. Therefore, in normal EIS practice, a small (1 to 10 mV) AC signal is applied to the cell: this is small enough to confine the test into a pseudo-linear segment of the cell's current versus voltage curve.
Second, the battery parameters are not constant: in general, even with open battery-terminals (i.e. zero current), the battery voltage varies over time depending on the previous history. To allow the stabilization of the battery voltage it is necessary to wait for the conditions of electrochemical balance in the battery. The required time, also referred to as "settling time" or "relaxation time" depends on the temperature (ion mobility) and is estimated in the order of a few hours. A measurement made before reaching the equilibrium condition produces data with variations especially in the lower part of the spectrum. These variations are more or less evident depending on the imbalance inside the cell.
Lastly, the voltage in a battery does not depend exclusively on the current flowing through it, but also on other parameters, in particular temperature and SoC. During each EIS measurement, these parameters must remain constant, in order not to influence the output voltage. In general, it must be ensured that the battery open circuit voltage does not vary within the range of the test, or this change will be computed in the spectrum of impedance.
Based on the assumptions above, the stimulus signal needed for the EIS test shall have the following characteristics:

- The spectrum of the stimulus shall adequately cover the whole frequency range that has to be analyzed (typically from 0.01 Hz to 1 kHz)
- The signal amplitude shall be "small enough" to avoid triggering any nonlinear response in the battery

As a drawback, the smaller is the signal amplitude, the worse is the signal to noise ratio. For this reason, a dedicated HW solution is needed to obtain a good resolution in the acquired signal. In particular, the proposed solution is:

- Amplify and adequately filter the signal, due to the small signal amplitude
- Remove the OCV voltage, which is not useful for EIS measurements
- Read each voltage value through a differential amplifier

All of the above-mentioned features are implemented in a dedicated EIS daughterboard, working together with the CCU-BMS board. Figure 6 below shows a functional block diagram of the daughterboard, integrating:

- The EIS Command Generator: a voltage DAC needed to generate the stimulus signal
- 12x EIS cell voltage measurement circuitry (differential amplifier + OCV cancelling + 4th order Bessel anti-aliasing filter)
- 2x EIS current measurement circuitry (4th order Bessel anti-aliasing filter)

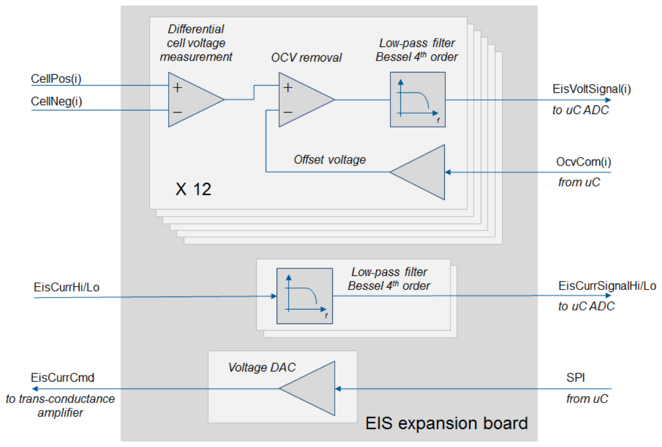


Figure 6: EIS daughterboard

In particular, the OCV has to be measured before applying the stimulus signal, and then removed through the dedicated DAC; this is part of the features implemented in the EIS software.

Finally, the stimulus signal generated by the daughterboard is a voltage signal; an external amplifier (in particular, a transconductance amplifier) is needed to drive the current that is injected into the cells. The chosen signal, used as a current stimulus, is a sum of sinusoidal current waveforms with predefined frequencies, in the range 0.01 Hz to 1 kHz, with a selectable current amplitude.

2) The resulting EIS algorithm

The EIS algorithm (Figure 7) injects a known current stimulus into the battery cell, reading the resulting voltage. Due to the assumptions described previously, the EIS algorithm will run after a relaxation time, needed for the battery parameters to reach a steady condition.

The measured signals from the battery shall then be analyzed, for each frequency, to determine the spectrum of the signal at that frequency. The idea is to correlate the measured signal to the input waveform, to obtain magnitude and phase information about the analyzed signal.

The response waveform from the battery typically has a DC offset, harmonic distortion components, and noise components generated by the cell. Nevertheless, the element of the measured signal, which needs to be analyzed, is the one at the same frequency as the generator waveform. All of the spurious components of the measured signal need to be rejected so that accurate measurements of the fundamental signal at the generator frequency can be made.

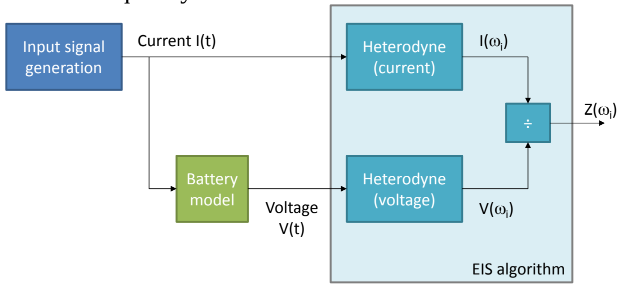


Figure 7: EIS algorithm overview

The measured system output is multiplied by both the sine and cosine of the test frequency ω. The results of the multiplications are then fed to two identical integrators, where they are averaged over T seconds. As the averaging time increases, the contribution of all unwanted frequency components go to zero and the integrator outputs become constant values which depend only on the gain and phase of the system transfer function at the test frequency.

Harmonics are rejected by the correlation process, and noise is rejected by averaging the signal over a number of cycles; the averaging associated with the correlating frequency response analyzer acts as a band pass filter with center frequency ω. As the average time T increases the bandwidth of the filter becomes narrower, thus the corrupting influence of wide band noise is increasingly filtered out as the correlation time is increased.

Averaging over a complete cycle avoids certain measurement errors associated with offsets on the system output; the performed simulations demonstrates that acquiring on a time window of three complete periods, we obtain an effective rejection of all frequencies above 0.1 Hz. Since the minimum frequency is 0.01 Hz, three complete periods corresponds to 300 seconds.

The result of the correlation process is made up of two components one of which is referred to as the *Real* (or in phase) component, the other is the *Imaginary* (or quadrature) component. By performing simple mathematical operations on these raw measurement results, it is possible to obtain the magnitude and phase of the impedance.

3) EIS Software implementation

The EIS Software consists of several SW components, using resources either directly from the Aurix microcontroller or through the EIS daughterboard

In particular, the EIS consists of:

- Complex Device Drivers (**CDD**):

  - OCV removal: generates the signals needed to remove the OCV from the measured cell voltage; OCV shall be measured before applying the stimulus signal, and then canceled through a dedicated DAC signal.
  - EIS Command Generator: its purpose is to generate the EIS Command signal (voltage reference) representing the current stimulus to be forced into the battery pack
- **iLLD** (Low-Level-Drivers from Infineon): mainly this will be used for the analog input signals acquisition and for the coherent measurement of:
  - Cell voltages - both DC and AC (useful EIS signal) - for each battery cell
  - EIS current flowing in the battery module/pack
- **Application**
  - Data Processing System for the calculation of the EIS spectrum

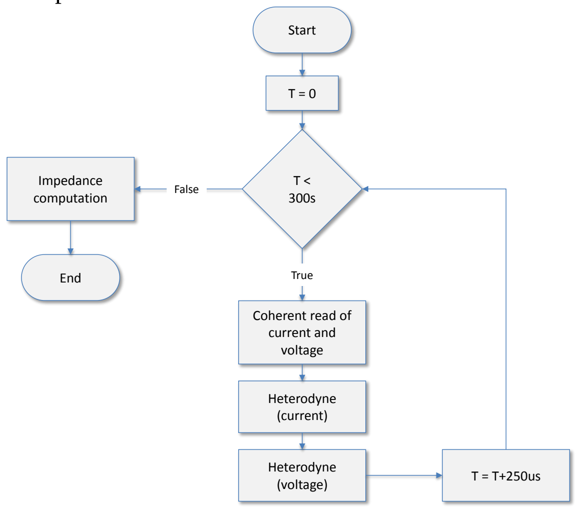


Figure 8: EIS application flowchart

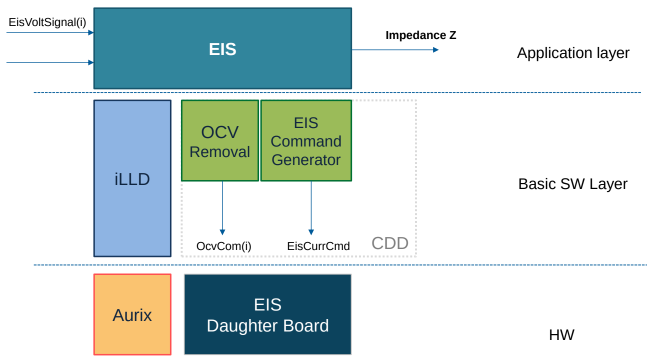


Figure 9: EIS software architecture

In particular, the application layer implements the EIS algorithm, as described above; the flowchart in Figure 8 shows an overview of the algorithm, that is executed for the whole time of the test (300 seconds), subsequently using then the last output from the integrators to compute the impedance. The resulting SW architecture is depicted in Figure 9.

4) Functional migration to parallel computing scheme

Since the EIS algorithm has been developed from scratch within the INCOBAT project, no specific migration from single core (sequential) to multi-core (parallel) computing scheme was required. The challenges are focused on the proper SW integration and resource managements. This will be discussed further in the next section.

IV. SOFTWARE INTEGRATION ON MULTICORE PLATFORM

A. Software development environment

The large variety of use cases as well as business organization is leading to different requirements on the development framework and build environment:

- Flexible configuration of source files, include files and directories for building code for each core. This targets increase in build efficiency as well as constructive integration [14] by the capability of updating a core independently from each other
- To have a sufficient intellectual property (IP) protection linking of external pre-compiled objects / libraries to the main binary of each core shall be possible. This is especially required in case of distributed development – means different teams / company integrating their IP into a common computing platform
- Adaptations to other compilers shall be possible with less effort.
- Integration of additional tools shall possible with less effort. These two last items are related to the distributed development of the entire SW by different teams relying on different development processes and consequently different tools

In the context of INCOBAT, the development environment shall be a low cost solution with capabilities to be deployed by each INCOBAT partner while minimizing the licensing costs. To meet all these requirements for the INCOBAT project a set of tools was used to establish the SW development framework.
The basic configuration of the SW development environment consists of a standard set of make files and target rules, a common memory mapping and compiler associated make and linker files.
To allow utilization of the ERIKA[2] operating system from Evidence into the integration and build process, the command line interface (CLI) from RT Druid was integrated in the build environment in form of make target rules. The OS configuration, including the definition and core allocation of counters, alarms, task, spinlocks and resources is done via OSEK implementation language (OIL) file, which is feed in RT Druid for the generation of the OS Erika related code and header configuration files for each core.
The build process is setup in such a manner that for each core a separate binary image is generated. This allows SW updates on one core without the need of rebuilding the other cores. Of course, this mechanism is only applicable if the applied changes do not affect the other's core SW and if the SW of the different cores can access peripherals of the microcontroller only via one dedicated interface. The SW code allocation to the different cores is done statically via one manual configurable make file, in which for each core application

[2] http://www.evidence.eu.com/

source and include files or directories, pre-compiled objects and libraries can be setup.

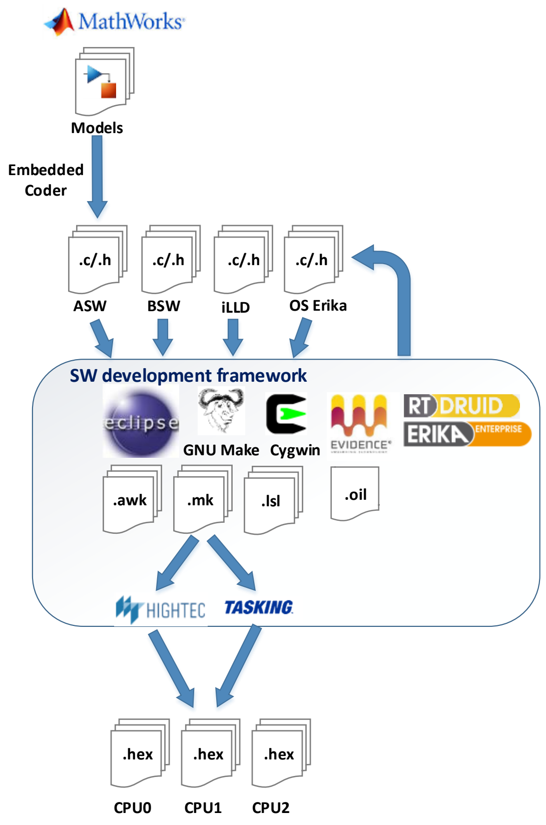


Figure 10: INCOBAT SW development framework

The memory allocation is done aligned to the AUTOSAR memory mapping approach and configured memory sections in the linker script. Depending on the currently identified CPUx in generation, the linker performs allocation of code and data to predefined flash and core local data scratchpad RAM memory sections.
The shared data of the cores is defined and allocated by the master core and placed into the local data scratchpad memory of that core which is the producer of the data element. The exchange of the memory information to the slave cores is done by dumping of master core's binary shared memory sections and export to a separate shared sub linker file which is generated during the build of the master core. The sub-linker script is used in the later build phases of slave cores for address resolutions of the shared data elements.
Another important aspect for the SW development environment was tool integration – in our case the mapping of the system information with the SW development framework. During the scope of the project, different tool interfaces for generating AUTOSAR aligned SW information where generated. The proposed tool interfaces mainly relies on four level of exchanges – all aligned with the AUTOSAR or OSEK standard. The first level (AUTOSAR tool-bridge) aims at describing the SW components (SW-C) and their interfaces – and serves proper integration of control strategies and application SW. The second level (RTE configuration) targets the description of the real-time environment for according configuration. The third level focuses on basic SW (BSW) configuration, while the fourth level aims at describing the operating system (tasks available in the system and their related options). More information is available in [15], [16] and [17].

B. *Software architecture*

To ensure modularity and reusability the SW architecture was split in several layers aligned to AUTOSAR:

- Infineon iLLD – similar to AUTOSAR MCAL, providing abstraction to HW I/O's, other peripheral modules, and startup code for the cores
- OS Erika – OSEK/VDX certified asymmetric operating system, where each core has its own copy of the OS instance
- BSW including complex device drivers and other coded BSW services
- ASW and ASWIL, to reduce the complexity of the system each ASW component is accessing its data via separate interfaces from the BSW or IOC module. During the execution of the function group, each core is using its local buffered data wherever possible to minimize execution time caused by inter-core accesses and remote blocking.

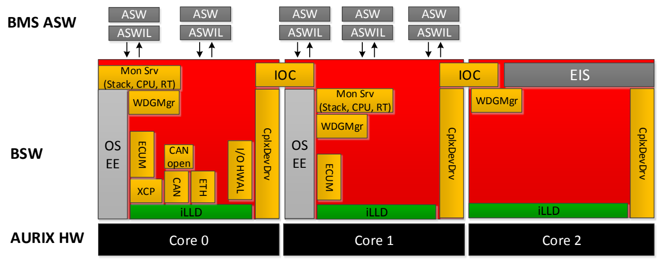


Figure 11: INCOBAT SW architecture

For the multicore capabilities, several SW functionalities were used similar to the currently defined and supported AUTOSAR concepts:

- Synchronized master slave startup and shutdown approach of the cores. During startup of master core 0 the other two slave cores are in idle. They startup with a synchronization barrier during startup of the OS. During shutdown, the reversed order is used and master core waits until synchronized shutdown of slave cores.
- Functional based inter-core data exchange of single signals or groups similar to the AUTOSAR Inter-OS-Applicator communicator (IOC).
- Usage of spinlocks to guarantee data consistency for core-to-core data exchange. The spinlock mechanism was combined with immediate suspension of interrupts in order to reduce the time of remote blocking. Additionally, to prevent from deadlocks nested acquisition of spinlocks was avoided.

C. *Verification environment – mini-HiL*

An important target and basis for the proper SW development, verification and integration is the deployment of appropriate test environment. Hence, the test environment shall be flexible enough to enable different kind of stimulation for the functions developed and realistic enough to accurately model the physics related to the system to control. In the context of INCOBAT, the simulation environment is playing an important role in different work-packages and tasks:

- White box testing: verification of single SW function such as control strategy (e.g., battery state estimation), safety function (e.g., control of battery's main relays) or basic SW (e.g., low-level drivers). Target is to provide the direct environment for these functions, therefore sometimes shortening the SW system by investigating only one function
- Grey box testing: validation of SW system and especially correct integration of the functions into the control system as well as correctness of the interfaces
- Black box testing: validation of the safety mechanisms – especially ensure correct reaction of the control system in case of hazardous situations

In the context of INCOBAT, different approaches are used:

- Model or SW in the loop (MiL / SIL): direct verification of single SW function
- Hardware in the loop (HiL): verification and validation of set of functions up to SW system in a real control system
- Vehicle demonstrator: prototyping validation in vehicle

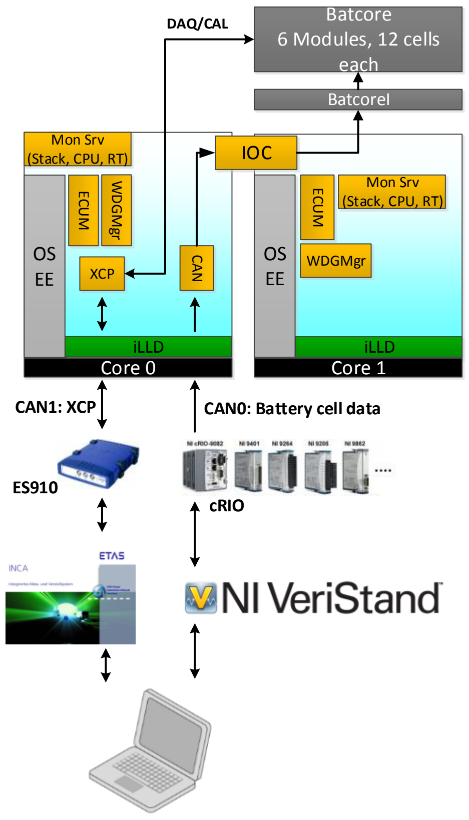


Figure 12: HiL test environment

While MiL and SiL are state of practice and will not be discussed any further, a dedicated mini-HIL platform for the efficient validation of the BMS (E/E system including satellite units and central controller) has been deployed. This platform is conjointly used for white, grey and black box testing – with different focuses as described previously,

For a validation of the SW multicore integration approach and the battery SoX estimation algorithms following HiL test environment was setup, see Figure 12.

In a first phase, the battery estimation algorithms were stimulated with battery cell data via CAN wrapper. For the stimulation of the battery load profiles the test and automation environment from NI Veristand was used in combination CompactRIO HW. With support of the XCP protocol, measurement and calibration access to each core's local memory was established.

D. Performances achieved and lessons learnt

A first comparison of the battery state estimation, while moving the computation accuracy from module to cell level, is shown in the following. The red curves represent the results of the modified estimation approach. Figure 13 illustrates the minimum and maximum cell SoC, while the discharge current limits during the cycle is illustrated in. Figure 14. Especially at the end of the cycle (at 6000s), a higher difference between the estimation on cell-level (red curve) and the estimation on module-level (blue curve) can be recognized. It is important to note that the results achieved in SiL and MiL environment are highly coherent (SoC and SoH difference below 0.05%) therefore confirming the correct integration into the multi-core computing platform.

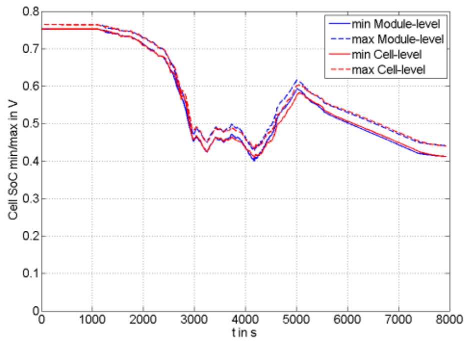


Figure 13: SoC computation at module and cell level

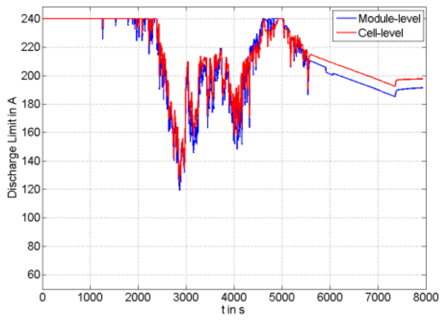


Figure 14: SoF computation at module and cell level

Figure 15 is summarizing the computing resource usage (task execution time) for the current implementation. As stated in Section 4-B, an instance of the operating system ERIKA is running on each core. Core 0 is managing the BSW and drivers, the battery state estimation algorithm is run on Core 1, and Core 2 is reserved for EIS. It can be noticed that the operating system task is consuming slightly more than 10% core time for this configuration. The computation of the SoX function at pack level requires less than 5% core time. By improving the computation accuracy to cell level, then a computation time of 15% of the core time is required. This illustrates the computing requirement (factor 3) in comparison to computation at module level. At the same time, it illustrates that integration of other functionalities into the AURIX platform (thus reducing the number of electronic control unit) is easily possible.

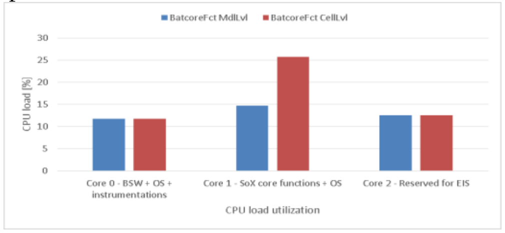


Figure 15: Computing resource usage for the three cores

It must be noted that at this stage of the project no particular approach was deployed for the systematic exploration of timing behavior and resource management. The decisions related to startup and shutdown sequences and handling, possible deadlocks, delays caused by remote blocking, memory allocation, as well as ASW functionality scheduling and partitioning were made based on expert knowledge. This step (scheduling analysis) will become essential to ensure an efficient and balanced functionality partitioning and scheduling and is part on ongoing work.

V. Conclusion

Accurate understanding and modeling of the physical behaviors of the cells' chemistry is pre-requisite for proper and optimized control of the HV battery, thus moving the limits for use of each single cell (and therefore of the entire battery at the end) from a conservative boundary to a more real limit. Consequently, it can be assumed that the range of the vehicle and cycle life of the battery can be increased due to the precise estimation approach.

The continuous advances in chip design (multi-core computing platforms for automotive applications) and embedded SW engineering (AUTOSAR) are providing important basis to deploy complex and accurate control strategies. At the same time, the competences and skills gap is growing apart between the different technologies. The seamless migration of an existing control strategy to a multi-core platform, while considering functional and non-functional requirements (e.g., performances, timeliness, safety), is not an easy task. During this paper, the migration of battery estimation functions (SoX) to an AURIX platform was presented. The migration has led to more accurate battery state estimation and illustrated that the proposed CPU provides enough performances for integration of further functionalities, thus providing the potential for reduction of number of discrete electronic control units within the vehicle. At the same time, an important lesson learnt was the need to proper analyze and manage startup and shutdown sequences, possible deadlocks, delays caused by remote blocking, memory allocation, as well as ASW functionality scheduling and partitioning. These aspects is already part of ongoing work.

Acknowledgment

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 608988

References

[1] G. E. Moore. Cramming more components onto integrated circuits. Electronics, 38(8):114 – 117, April 1965.

[2] Eric Armengaud, Georg Macher, Can Kurtulus, Riccardo Groppo, Sven Haase, Günter Hofer, Claudio Lanciotti, Alexander Otto, Holger Schmidt, Slawomir Stankiewicz, Improving HV battery efficiency by smart control systems, Smart Systems Integration conference 2015 (SSI 2015)

[3] Brewerton, Schneider, & Grosshauser. (2009). Practical use of AutoSAR in Safety Critical Automotive Systems. SAE #2009-01-0748.

[4] Schneider, Eberhard, & Brewerton. (2010). Multicore vs. Safety. SAE #2010-01-0207.

[5] ISO 26262: "Road vehicles – Functional safety", 2011.

[6] W. Waag, C. Fleischer and D. U. Sauer, "Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles," in Journal of Power Sources 258, 2014.

[7] G. L. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs Part1. Background," Journal of Power Sources, pp. 252-261, 2004.

[8] G. L. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 2. Modeling and identification," Journal of Power Sources, pp. 262-276, 2004.

[9] G. L. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 3. State and parameter estimation," Journal of Power Sources, pp. 277-292, 2004.

[10] G. L. Plett, "Sigma-point Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 1. Introduction and state estimation," Journal of Power Sources, pp. 1356-1368, 2006.

[11] G. L. Plett, "Sigma-point Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 2: Simultaneous state and parameter estimation," Journal of Power Sources, pp. 1369-1384, 2006.

[12] G. L. Plett, "Battery management system algorithms for HEV battery state-of-charge and state-of-health estimation," in Advanced Materials and Methods for Lithium-Ion Batteries, 2007.

[13] B. Kortschak, C. Kurtulus, M. Dohr, U. Wiedemann and V. Hennige, "Detection Method of Battery Cell Degradation," in Vehicle Power and Propulsion Conference, Chicago, IL, 2011.

[14] Hermann Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications". Kluwer Academic Publishers,, Apr 2011, ISBN 978-1441982360

[15] G. Macher, E. Armengaud, and C. Kreiner. Integration of Heterogeneous Tools to a Seamless Automotive Toolchain. In R. O'Connor and R. Messnarz, editors, EuroSPI 2015, 2015.

[16] G. Macher, M. Atas, E. Armengaud, and C. Kreiner. A Model- Based Configuration Approach for Automotive Real-Time Operating Systems. SAE International Journal on Passenger Cars – Electronics and Electronical Systems, 8(2), 2015.

[17] G. Macher, R. Obendrauf, E. Armengaud, and C. Kreiner. Automated Generation of Basic Software Configuration of Embedded Systems. In ACM RACS Conference Proceedings, 2015.

Session 23

Static Analysis

Thursday 28th, 16:30 – Ariane 1

Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée

Antoine Miné[2], Laurent Mauborgne[5], Xavier Rival[1,3], Jerome Feret[1,3], Patrick Cousot[4], Daniel Kästner[5], Stephan Wilhelm[5], Christian Ferdinand[5]

[1]École normale supérieure, Paris, France
[2]Sorbonne University, University Pierre and Marie Curie, CNRS, LIP6
[3]INRIA, France
[4]Courant Institute of Mathematical Sciences, NYU, New York
[5]AbsInt GmbH, Saarbrücken, Germany, http://www.absint.com

Abstract

We present an extension of Astrée to concurrent C software. Astrée is a sound static analyzer for run-time errors previously limited to sequential C software. Our extension employs a scalable abstraction which covers all possible thread interleavings, and soundly reports all run-time errors and data races: when the analyzer does not report any alarm, the program is proven free from those classes of errors. We show how this extension is able to support a variety of operating systems (such as POSIX threads, ARINC 653, OSEK/AUTOSAR) and report on experimental results obtained on concurrent software from different domains, including large industrial software.

1 Introduction

Safety-critical embedded software has to satisfy stringent quality requirements. All contemporary safety standards require evidence that no data races and no critical run-time errors occur, such as invalid pointer accesses, buffer overflows, or arithmetic overflows. Such errors can cause software crashes, invalidate separation mechanisms in mixed-criticality software, and are a frequent cause of errors in concurrent and multi-core applications.

The last years have seen the emergence of semantics-based static analysis tools able to detect run-time errors, such as the Astrée analyzer [6]. However, such tools cannot handle concurrent programs at all, or with the same level of soundness, coverage, and automation as sequential programs: they would not cover all potential process interleavings, or require the user to enter manually the set and range of shared variables, or miss support for concurrency primitives (such as mutexes) or the detection of concurrency-specific hazards (such as data races). We present here an extension of Astrée to analyze soundly and automatically concurrent software.

The article is structured as follows: first, in Sec. 2, we give an overview of sound static analysis and of Astrée. In Sec. 3, we explain the key concepts underlying our interleaving semantics, which makes it possible to analyze concurrent programs in a scalable and sound way, and report all run-time errors and data races. Section 4 discusses our support for several standard operating systems, enabling the automated analysis of software running under these OS. Section 5 discusses our experiments: the analysis of industrial avionic software, as well as preliminary results on ongoing experiments on OSEK software. Section 6 discusses related work. Section 7 concludes.

2 Overview of Astrée

Sound static analysis. Astrée discovers errors by inspecting the source code without running it. It traverses the program control structure and interprets program instructions according to the language semantics to build automatically a model of its executions. To ensure efficiency, the model must be approximated, but we take care to always use over-approximations. Thus, in contrast to most other static analyzers, Astrée makes sure that all possible program executions are taken into account: it achieves a full coverage of the whole control and data space of the program. For this reason, it is sound: whenever no error is reported, we are certain that no error can exist in the actual program executions either. Like all sound static analyzers, Astrée may report false alarms (notifications about potential run-time errors which do not occur in real program executions). An important design goal of the analyzer, reached in 2003, was to achieve zero false alarm on a significant class of sequential software: large industrial avionics control-command software [6].

Language. Astrée has been developed for safety-critical C programs and is based on the C99 standard [17]. It supports all C control structures and C datatypes, provides a stubbed C library and even supports dynamic memory allocation. The only notable limitations are that recursive calls will be detected and reported as an alarm without trying to analyze the recursive invocations; moreover, long jumps are not supported. As a recent extension, Astrée supports concurrency features, such as threads and locks (Sec. 3) and can analyze software running on top of operating systems implementing common standards (e.g., POSIX, ARINC 653 [4], OSEK/AUTOSAR [1], see Sec. 4).

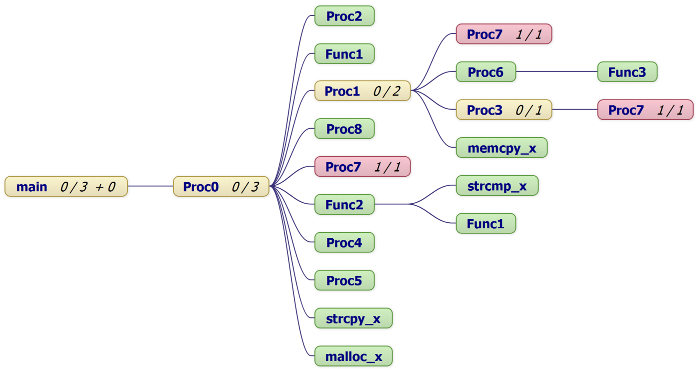
Semantics. The semantics of programs used by Astrée is based on the C99 standard [17]. However, the standard provides a high-level and abstract semantics, leaving many aspects of program behaviors implementation-defined, unspecified, or undefined. Implementation-defined features, such as the bitsize of integers can be configured. Moreover, Astrée employs a low-level memory semantics, which is aware of the bit-level representation of objects, that can be configured as well [21] (cf. also Sec. 3). This gives a semantics to undefined behaviors, such as type punning or wrap-around after signed arithmetic overflow, often used in low-level embedded code, and allows Astrée to analyze such programs correctly and precisely. This low-level semantics also frees Astrée from the reliance on static type information, so that it can handle the common case where an unstructured array of bytes is dynamically reinterpreted as a structure of some type. Astrée can also handle multi-dimensional arrays encoded explicitly in a single array using index arithmetic. Floating-point numbers are modelled faithfully according to the IEEE 754 norm [15], including special numbers (infinities and *NaN*) as well as rounding.

Error checking. Astrée signals all potential runtime errors and further critical program defects. It reports program defects caused by unspecified and undefined behaviors according to the C99 standard [17], program defects caused by invalid concurrent behavior, violations of user-specified programming guidelines, and computes program properties relevant for functional safety. Astrée raises alarms for operations resulting in unpredictable program behaviors, such as invalid array and pointer accesses. Alarms are also raised for invalid operations triggering exceptions, such as divisions by zero or floating-point overflows, and for dangerous operations whose result, although well-defined either in the C99 standard or in Astrée's more refined semantic, may be unexpected, for instance wrap-around after unsigned or signed arithmetic overflow. Astrée does not stop at the first error, but strives to continue the analysis with a reasonable result. This is useful to handle, e.g., programs with intended wrap-around, and prevents a benign error from masking a subsequent, more serious one. Astrée also permits users to specify their own functional properties to be checked with an assertion mechanism (similar to C's assert command), and will report any violation. Finally, Astrée includes a rule checker that supports MISRA C:2004 [26] and MISRA C:2012 [27] and can be extended for customer-specific rule sets.

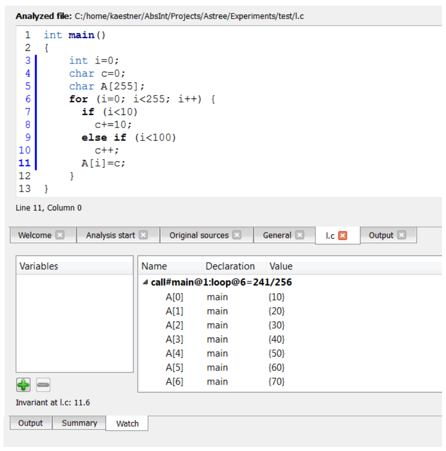
Abstraction. Astrée is based on abstract interpretation [7]: it uses abstractions to represent and manipulate efficiently over-approximations of program states. One simple example of abstraction used pervasively in Astrée is to consider only the bounds of a numeric variable, forgetting the exact set of possible values within these bounds. However, more complex, but also more costly, abstractions can also be necessary, such as tracking linear relationships between numeric variables (which is useful for the precise analysis of loops). As no single abstraction is sufficient to obtain sufficiently precise results, Astrée is actually built by combining a large set of efficient abstractions (e.g., the octagon domain [22]). Some of them, such as abstractions of digital filters [10], have been developed specifically to analyze control-command software as these constitute an important share of safety-critical embedded software. In addition to numeric properties, Astrée contains abstractions to reason about pointers, pointer arithmetics (abstracting offsets as numeric variables), structures, arrays (in a field-sensitive or field-insensitive way). Finally, to ensure precision, Astrée keeps a precise representation of the control flow, by performing a fully context-sensitive, flow-sensitive (and even partially path-sensitive) interprocedural analysis.

Analysis options allow fine-tuning the analysis precision, either with global parameters or with local directives focusing precision on some program parts and some variables. All Astrée directives, e.g., for specifying range information for inputs or adapting the precision of the analyzer can be specified in the formal language AAL [3] by locating them in the abstract syntax tree without modifying the source code — a prerequisite for analyzing automatically generated code. To deal with evolving software Astrée provides a mechanism to detect whether annotations are still placed at the intended location after structural code changes [19].

Analysis output. In addition to the list and location of alarms, Astrée makes the semantic information computed during the analysis available to the user. For instance, Astrée constructs, based on its analysis of function pointers, a control-flow graph, which can be visualized graphically and interactively explored after the analysis. Furthermore, the range computed for each variable, at each location and for each call context, can be looked-up. This provides additional useful information about the program: it can be used, beyond run-time error checking, to verify design specifications. It is also



(a)



Analyzed file: C:/home/kaestner/AbsInt/Projects/Astree/Experiments/test/l.c

```
1  int main()
2  {
3      int i=0;
4      char c=0;
5      char A[255];
6      for (i=0; i<255; i++) {
7        if (i<10)
8          c+=10;
9        else if (i<100)
10         c++;
11       A[i]=c;
12     }
13 }
```

Line 11, Column 0

Name	Declaration	Value
call#main@1:loop@6=241/256		
A[0]	main	{10}
A[1]	main	{20}
A[2]	main	{30}
A[3]	main	{40}
A[4]	main	{50}
A[5]	main	{60}
A[6]	main	{70}

Invariant at l.c: 11.6

(b)

Figure 1: Astrée call graph visualization (a) and variable range (b) visualization.

useful for alarm investigation, to understand the origin of run-time errors and spurious alarms. Astrée also reports unreachable code and non-terminating loops.

3 Concurrent Program Analysis

Astrée has been recently extended [23] to support concurrency-related constructions, with a specific focus on concurrency features for embedded C software. Traditionally (prior to the C11 standard, which includes concurrency into the C language, but even after), concurrency is provided to a C implementation through additional libraries, with varying, incompatible semantics. To solve this issue, Astrée provides universal low-level building blocks for concurrency features, on top of which realistic models of actual concurrency libraries can be programmed. This section focuses on the semantics and analysis of the low-level concurrent semantics, while concurrency library modelling is discussed in Sec. 4.

Threading model. Astrée's low-level concurrency semantics is based on POSIX-style threads [16]. Each thread is a fully preemptable execution unit with independent control and local variables, but shared global memory. A program execution is then an interleaving of thread executions. Thus Astrée threads can be used to model POSIX threads, but also ARINC 653 processes, OSEK/AUTOSAR tasks, interrupts, etc. Depending on the concurrency model, threads may be declared in an external configuration file (such as OSEK tasks) or programmatically (as in POSIX threads). Astrée supports both models, but assumes in the latter case that threads cannot be created arbitrarily during program execution. Instead, program execution is decomposed into two separate phases: an initialization phase that executes arbitrary sequential C code and can create, but not execute, threads; and a second phase where all threads execute concurrently but new threads cannot be created. This limitation matches the current practice (and sometimes the OS limitations) in embedded software. It is exploited to achieve a simpler and more precise analysis. The set of threads created programmatically is discovered during the analysis fully automatically. Additionally, Astrée supports the concept of thread instances, i.e., multiple creations of threads with the same entry point. The thread-modular abstraction used in Astrée, described below, reduces the analysis of the program to that of a single instance of each thread. Hence, Astrée naturally supports unbounded instances of threads, which is useful to analyze parameterized systems, i.e., systems where the number of instances of a thread is an unknown constant.

Shared memory. Following the POSIX thread model, Astrée assumes that all the threads can access all the global variables, i.e., the global variables are implicitly shared. By analyzing the threads, Astrée then infers automatically which variables are actually shared and reports precisely which part of each variable is accessed by each thread and the access mode (read, write or read/write). While it is possible for one thread to access the local variables of another thread (e.g., sharing a pointer to a local variable through a global variable) this is a dangerous practice as the local variables can be deallocated by the time the other thread accesses it. Astrée thus detects and reports such usages as errors. Similarly, while Astrée supports dynamic memory allocation (e.g., with `malloc`), it is an error (reported to the user) for one thread to access the memory allocated by another thread.

Synchronization. Astrée has built-in support for thread synchronization. In particular, Astrée has a notion of mutual exclusion locks, so-called mutexes, with the property that a given mutex can be locked by at most one thread at a time. Astrée's mutexes are very simple non-recursive variants of POSIX's mutexes: if a thread locks a mutex that is locked by anther thread, it enters a waiting state until the other thread unlocks the mutex; locking again a mutex that is already locked by the same thread, or unlocking a mutex that the thread has not locked, has no effect. More complex locking mechanisms can be programmed on top of such simple mutexes to model the semantics of realistic concurrency libraries (such as recursive mutexes that feature a lock counter, or mutexes that fail when locked again by the same thread). In Astrée, mutexes are identified by 32-bit integers and need not be created *a priory*. It is the responsibility of the OS modelling (Sec. 4) to allocate such integers, either statically (e.g., associate a mutex to each resource in an OSEK program) or programmatically (e.g., use a counter to allocate at run-time unique mutex identifiers when a new mutex is created by a POSIX thread system call).

Astrée tracks which part of each thread is protected by each mutex, and discovers automatically regions that are in mutual exclusion. This information is combined with the inference of shared memory locations, so that Astrée can report all data races (both read/write and write/write data races). In case of a data race, Astrée continues the analysis by considering the possible values steaming from all possible interleavings.

producer

```
for (i=0;i<100000;i++)
{
  lock(1);
  x=x+1;
  if (x>100) x=100;
  unlock(1);
}
```

consumer

```
for (j=0;j<100000;j++)
{
  lock(1);
  if (x>0) x=x-1;
  unlock(1);
}
```

Figure 2: Producer and consumer threads protected by a mutex.

Example. Figure 2 gives an example program composed of one or several instances of a producer thread and one or several instances of a consumer thread, where the resource is abstracted as a counter variable x. In this example, Astrée will be able to discover that x is shared and that there is no data race, as all the accesses to x are correctly protected by mutex 1. Additionally, Astrée reports that x is always in the range $[0, 100]$, except just after `x=x+1`, where it can be 101. Failure to use a mutex would cause Astrée to report a data race at each access to x. It would also cause the range of x to grow beyond 101 as several producer instances can now concurrently increase x before the test `if (x>100) x=100`.

Astrée does not currently detect deadlocks caused by improperly nesting of mutex locks by concurrent threads. This is not an inherent limitation of our method, but a limitation of the tool, and this detection is planned for future work, by leveraging the automatic detection of which mutexes are locked by each thread at each program point. Additionally, Astrée has a preliminary support for additional synchronization primitives: read/write locks, signals, and barriers, which are currently handled in a sound but sometimes imprecise way, and future work to improve their support is planned.

high priority

```
for (i=0;i<100000;i++)
{
  if (!islocked(1))
  {
    x=x+1;
    if (x>100) x=100;
  }
  yield();
}
```

low priority

```
for (j=0;j<100000;j++)
{
  lock(1);
  if (x>0) x=x-1;
  unlock(1);
}
```

Figure 3: Priority-based producer-consumer example.

Real-time scheduling. Astrée is sound with respect to all possible interleavings of threads, which would correspond to a fully preemptive and non-deterministic scheduler. However, embedded programs often employ specific real-time schedulers that partially restrict thread interleavings. Notably, each thread is given a priority, and higher priority threads cannot be preempted by lower priority ones, unless they stop explicitly by issuing a blocking system call, such as locking a mutex or waiting for an external event. Astrée takes priority information into account, when available, to detect portions of threads in mutual exclusion due to priority scheduling, and it uses this information to remove spurious thread interactions and data races.

Example. Figure 3 presents a variant of Fig. 2 using priorities. After testing whether the mutex is unlocked, the high priority thread can assume that the low level priority thread is not in its critical section; it can then safely test and modify x atomically, without fear of being interrupted by the low priority thread. The effect is thus the same as in the program of Fig. 2. Astrée proves the absence of data race and provides precise bounds for x.

Note that, at the end of its critical section, the high priority thread explicitly yields to allow the lower priority thread to run. The semantics of the `yeild` primitive is that of a non-deterministic wait, which is useful to model waiting for an external event or for a delay (as Astrée does not keep track of execution time). As a consequence of this non-determinism, the high priority thread may interrupt the lower priority thread at *any* point during its execution. This highlights the fact that, despite a deterministic, priority-based scheduling, embedded programs often feature a large possible number of thread interleavings. Unlike previous works on embedded real-time applications [11], Astrée is not limited to collaborative threads, nor discrete sets of preemption points, which would not soundly account for all possible executions. Note that, to ensure scalability, Astrée employs possibly imprecise abstractions of thread priorities and real-time scheduling. For instance, threads with dynamically changing priorities are supported, but considered to be preemptable by all threads at any point (i.e., their exact priority relative to other threads is not tracked), which is sound but imprecise. To improve precision we are currently implementing the priority ceiling protocol which is the standard scheduling scheme in OSEK systems. When unable to use priorities to reduce the interleaving space, Astrée reverts to unrestricted preemption, which ensures a coverage of all concurrency models.

Thread-modular analysis. On sequential programs, Astrée employs a fully flow-sensitive and context-sensitive analysis: an abstraction of the possible memory states is propagated along the program control flow graph, and abstract states are merged at control-flow joins (such as the end of an if-then-else or a loop iteration). Flow-sensitivity, i.e., the ability to distinguish the value of a variable at different control points, is often necessary to achieve a degree of precision sufficient to prove the absence of run-time error. Concurrent programs, however, feature a far more complex control structure than sequential ones, which makes it unpractical to consider a fully flow-sensitive analysis. There is a combinatorial explosion of the number interleaved execution paths and it would be too costly to distinguish the value of a variable at each combination of thread control locations.

For concurrent programs, Astrée thus employs instead a thread-modular analysis. In a nutshell, each individual thread of the program is analyzed separately, as would be a sequential program. In addition to potential run-time errors, each thread analysis collects the effect it can have on the global memory. The threads are then reanalyzed, but now taking into account the effect from other threads as gathered at the previous analysis. As this new analysis may expose new behaviors of threads, and so, more effects, it triggers a reanalysis of the threads. The analysis thus proceeds in rounds, starting from an empty set of thread interactions, and reanalyzing the threads with an increasing interaction set, until stabilization. A standard abstract interpretation technique, iteration extrapolation with widening, is used to ensure that this process terminates after a finite, small number of iterations (experiences point towards around 6 iterations, independently from the program size and number of threads). A theoretical result [23] states that, after stabilization, the thread-modular analysis has explored an over-approximation of all the possible interleavings; it is thus sound.

Example. Consider again the producer-consumer example from Fig. 2. The first analysis round, considering each thread in isolation, will deduce that, at the end of the producer loop, `x` necessarily equals 100 while, at the end of the consumer loop, `x` necessarily equals 0, which is obviously inconsistent. However, the analysis also deduces that, during its execution, the producer stores a value in $[1;101]$ into `x`, and the consumer does not modify `x` (yet). This information is used at the second analysis round. In particular, now, when the consumer performs `x=x-1`, this is understood as storing into `x` the last value stored into `x` by the consumer minus 1, or storing a value stored by the producer, i.e. $[1;101]$, minus 1. The analysis of mutexes further deduces that the value 101 is not actually visible by the consumer, hence the second case stores a value in $[1,100]-1=[0,99]$ into `x`. At the end of the consumer loop, `x` would thus read either a value in $[0,99]$, when reading the last value stored by the consumer, or a value in $[1,100]$, if a write from the producer was performed since that last write by the consumer. A third analysis round, where the consumer takes into account the values $[0,99]$ stored by the consumer, yields the same set of interferences, hence, the analysis finishes and deduce that, at the end of the program, `x` is in the range $[0,100]$, which is the expected result.

The benefit of this method is threefold. Firstly, it provides a sweet spot between cost and precision: it is nearly as efficient as a sequential program analysis and maintains flow-sensitivity at the intra-thread level. Secondly, each thread analysis is but a sequential program analysis, slightly modified to extract and apply interferences on the shared memory; thus, all the infrastructure present in sequential Astrée could be reused as is. Thirdly, the analysis is parametric independently in the abstraction chosen to abstract the memory and the abstraction chosen to abstract thread interferences. The former exploits all the memory abstractions developed for sequential Astrée. For the later, the above example employs a simple and scalable, non-relational and flow-insensitive abstraction: the range of values stored by a thread into a variable, but recent work [24] has proposed new abstractions that can improve the precision without sacrificing the scalability by adding a small measure of relationality or flow-sensitivity; Astrée is thus able to infer that a thread modifies a variable in a monotonic way, and to discover relational locks invariants.

Memory consistency. When several threads access a shared memory, it is important to determine the underlying consistency model ensured by the hardware and compiler. The simplest model, *sequentially consistent memory* [20], assumed implicitly in our examples above, states that, in an interleaving of thread executions, each thread reads back from the shared memory the value stored by the last thread to write into the memory. This is unfortunately not realistic: modern hardware introduce memory hierarchies, buffers and cache, and compilers introduce optimizations that invalidate this view, as several copies of a variable may reside in the system. Modern language specifications, such as C11, introduce weaker memory models to take such effects into account. As weak memories feature non sequentially consistent executions, an analysis tool designed solely for sequential consistency is not sound with respect to a weak memory model. In contrast, Astrée is designed to be sound for a variety of memory models, based on the choice of which abstractions are used for thread interferences. For instance, the flow-insensitive non-relational abstraction used in the above example has been proven [23] to be sound for very lax memory models, while the soundness of the abstraction able to infer the monotonicity of shared variables requires a model such as *total store ordering* adopted by several popular processors, such as x86 [32].

4 Operating System Support

Programs to be analyzed are seldom run in isolation; they interact with an environment. In order to soundly report all run-time errors, Astrée must take the effect of the environment into account. In the simplest case (e.g., the most critical software), the sofware runs directly on top of the hardware, in which case the environment is limited to a set of *volatile variables*, i.e., program variables that can be modified by the environment concurrently, and for which a range can be provided to Astrée by formal directives. More often, the program is run on top of an operating system, which it can access through function calls to a system library. When analyzing a program using a library, one possible solution is to include the source code of the library with the program. This is not always convenient (if the library is complex), nor possible, if the library source is not available, or not fully written in C, or ultimately relies on kernel services (e.g., for system libraries). An alternative is to provide a *stub* implementation, i.e., to write, for each library function, a specification of its possible effect on the program.

Library stubs. Astrée provides facilities to concisely write stubs that model functions at an abstract level using C code with additional primitives, including non-deterministic variable modifications and checked assertions (using arbitrary C boolean expressions). A typical stub first checks the validity of its arguments (using assertions), then performs necessary side-effects (such as modifying an argument passed by reference) and finally constructs a valid return value. For instance, the `sin` stub function only checks that its argument is a not a special floating-point number and returns a non-deterministic value assumed only to be in $[-1,1]$. Astrée comes with a complete set of stubs for the C library, weighting 9 Klines. It is based on the C99 standard [17], not on a specific implementation; as a result, the analysis results are sound whatever conforming C library implementation is used.

Concurrency stubs. With the addition of concurrency, new libraries have been added, including POSIX threads [16] and the ARINC 653 standard used in avionics [4]. These leverage the low-level concurrency primitives offered by Astrée and its internal notion of threads and mutexes, but often need to wrap them into more complex objects maintained in C arrays and structures. For instance, a POSIX thread is an Astrée thread together with attributes and a state (such as a cleanup routine, a return value, etc.). Additionally, the core set of Astrée objects is reused to model the wide variety of objects offered by such systems; e.g., asynchronous signal handlers are assigned an Astrée thread, mutexes are reused to implement read-write locks, etc. Around 3 Klines of the 9 Klines of C library are devoted to POSIX concurrency primitives, while the model of ARINC 653 occupies 4 Klines. More details on these models are available in [25].

OSEK/AUTOSAR support. Astrée has recently added support for OSEK/AUTOSAR operating systems [1], a widely used standard in automotive. An OSEK/AUTOSAR program consists of a set of tasks, a set of interrupts (also called ISRs), a set of timers (also called alarms), and schedule tables (a data-driven mechanism to activate tasks). Task scheduling and synchronization is achieved through explicit task activation and chaining, the use of priorities, orders to disable and enable interrupts, the use of resources objects (that act as locks), and events (that act as signals).

We provide an OSEK/AUTOSAR library that handles these mechanisms by mapping them to Astrée low-level concurrency objects: tasks, ISRs, alarms and schedule tables are mapped to Astrée threads; resources are mapped to Astrée mutexes; events are mapped to Astrée signals; moreover, Astrée natively supports the relevant notions of priorities and offers built-in primitives to achieve chaining, starting, and stopping. Note that, due to the abstractions employed by Astrée to achieve scalability, some aspects of scheduling are not currently analyzed in a precise way. For instance, Astrée does not currently track which threads are in a stopped or started state, and assumes that every thread is possibly started at any point. As a result, interrupts enable and disable operations are not precisely handled. We plan to address this limitation in future work by simply adding new abstractions without changing the model.

The standard proposes several conformance classes, with support for increasingly complex features (such as extended tasks, fully preemptive scheduling, multiple task activation, etc.). The model proposed in Astrée supports the most general class, which guarantees that all programs can be soundly analyzed.

A particularity of OSEK/AUTOSAR is that all system resources, including tasks, are not created dynamically at program startup. Instead they are hardcoded into the system: a specific tool reads a configuration file in OIL format describing these resources and generates a dedicated version of the system to be linked

Size	Added	Select.	Time	Mem.
2.1 M	5.2 K	99.94%	24 h	27 GB
1.9 M	2.4 K	99.56%	154 h	18 GB
2.2 M	2.3 K	99.52%	160 h	23 GB
31.8 K	2.2 K	97.28%	50 mn	0.6 GB
33.1 K	1.2 K	97.18%	35 h	2.5 GB

Figure 4: Avionics case studies from [25], with the original size (in lines), the size (in lines) of added stubs, the selectivity (percentage of lines proved correct), the analysis time and memory consumption.

against the application. Astrée supports a similar workflow. In the preprocessor stage it can read OIL files and outputs a C file containing a table of the declared resources, with their attributes (task priority, alarm periodicity, etc.). The OIL file also assigns actions to be executed when an OSEK alarm expires, such as activating a given task or event, or calling a call-back. The preprocessor thus generates specific C functions to handle the actions associated to OSEK alarms. A fixed set of application-independent stubs, comprising 3 Klines of C with Astrée directives, implements the 31 OSEK entry points. The fixed stub also contains a main analysis entry point that creates Astrée threads and mutexes according to the generated tables and enters parallel execution mode. Finally, it contains synthetic entry-points for Astrée threads handling OSEK alarms, whose purpose is to call, at non-deterministic intervals, the functions generated by the preprocessor to implement the actions associated to OSEK alarms. Combining the C sources of the OSEK application, the fixed OSEK stub provided with Astrée, and the C file automatically generated from the OIL file, we get a stand-alone application, without any undefined symbol, that can be analyzed with Astrée and models faithfully the execution of the application in an OSEK environment. This workflow enables a high level of automation with minimal configuration when analyzing OSEK applications.

The set of errors detected by Astrée includes runtime errors and data-race, but also a new alarm category *invalid usage of OS service*. As an example the OSEK stub automatically checks that the application calls OSEK services according to the specification. In case of API errors the analysis of an OSEK application will raise alarms from this new category, including: invalid task, alarm, or resource identifiers, calling a service from an ISR with incorrect level, improper nesting of resource acquisition and release (lock/unlock problems), or failure to release all the acquired resources before terminating a task.

5 Practical Experiments

The concurrency support built into Astrée has been tested in a variety of analysis experiments.

Name	Size	Select.	Time	Mem.
HiTechnic	162	100%	0.4 s	11 MB
NXT GT	302	97%	1.2 s	20 MB
NXTway-GS	439	98%	4.1 s	20 MB
NXT Cesar	4500	95%	6 mn	435 MB

Figure 5: Preliminary OSEK experiments on nxtOSEK samples [2, 14].

5.1 Avionics Software – ARINC 653

The support for ARINC 653 was first designed as a research experiment extending Astrée to analyze medium-sized to large concurrent industrial avionics C software. Astrée was later extended with a subset of POSIX threads, also used in avionics software. The results of these experiments are reported in details in [25] and summarized in Fig. 4. To sum-up, this study shows that Astrée can handle complex, realistic concurrent programs with a sufficient level of precision (a selectivity near 100%, indicating that very few lines exhibit an alarm) and adequate performance in the context of software validation (where tests, the usual validation method, can take weeks).

5.2 Automotive Software – OSEK

In the following we summarize experimental results obtained on OSEK applications: some small C programs designed for Lego Mindstorm NXT robots under the nxtOSEK system [2], and three real-life automotive applications. For reasons of confidentiality the results on industrial automotive projects have been anonymized. The results show that Astrée can be successfully applied on real-life industrial software projects. Moreover, the analysis runs on standard PC hardware and is reasonably fast.

Lego Mindstorm. As a proof-of-concept, our initial tests of the OSEK support in AstréeA were performed on simple, freely available C programs designed for the Lego Mindstorm OSEK platform. The results are shown in Fig. 5. The first three programs, of a few hundred lines, are sample programs included in the nxtOSEK distribution. The last program is the NXT Cesar robot developed at the iCube laboratory [14]. This program performs non-trivial floating-point computations, on which Astrée reports possible overflows and invalid operations; indeed the software elects to perform computations without checking operator arguments, and fix the result only after the computation, by replacing any infinity and not-a-number with zero.

Automotive 1. The first real-life application is a small project consisting of two tasks comprising 177 576 lines of preprocessed C code (without blank lines and without comments). The project is configured by an `.oil` file automatically processed by Astrée. Astrée reports 698 alarm locations with alarms of the following type:

Alarm Category	#Loc
Invalid range of pointers and arrays	17
Division or modulo by zero	58
Invalid ranges and overflows	617
Read/write data race	6

The analysis takes 38min with full precision and consumes 7.5 GB RAM. It reaches 78% of the code. The 6 alarms about read/write data races were all confirmed to be justified, there were no false alarms about data races.

Automotive 2. The second real-life application consists of 358 335 lines of preprocessed C code (without blank lines and without comments). The configuration is given by an `.oil` file which can be automatically processed by Astrée to produce all relevant data structures and access functions. The project consists of 4 tasks, 30 ISRs (interrupts) and 3 alarms (timers). Astrée reports 1 796 code locations with alarms of the following types:

Alarm Category	#Loc
Division or modulo by zero	58
Invalid usage of pointers or arrays	460
Invalid ranges and overflows	1 278

With reduced precision settings the analysis reaches 97% of the code, the analysis time is 7h13min, and memory consumption 3.1GB. The resulting selectivity is above 99%; these alarms include run-time errors caused by the effects of data races, e.g., overflows or invalid pointer accesses, but not the data races itself. This distinction is reasonable since there may be data races which do not actually induce erroneous behavior.

Furthermore Astrée reports 15 967 code locations with alarms from the newly introduced concurrent alarm categories:

Invalid Concurrent Behavior	#Loc
Read/write data race	8 024
Write/write data race	7 941
Invalid usage of OS service	2

A data race alarm is produced for every access contributing to a race, i.e. for each shared variable subject to a data race several alarms will be issued. This increases the number of of alarms reported but helps users to distinguish between correct accesses and accesses contributing to a race. A further analysis of the data races shows that in this project most of the synchronization is done via explicit enable/disable interrupt calls. Currently such calls are not precisely handled, but dedicated abstractions for them are under development. Also task priorities have been exploited to optimize the application: write operations are mostly done by the highest-priority task which enables light-weight synchronization mechanisms. As explained in Sec. 6 the support of the priority ceiling protocol currently is in development, too, so to enable a sound result, task priorities are currently not taken into account. With both extensions finished we expect the number of data race alarms to be significantly reduced.

Automotive 3. The third real-life project is an OSEK application with 1 655 384 lines of preprocessed C code (without blank lines and without comments), again configured by an `.oil` file. The project consists of 24 tasks, 34 ISRs and 12 alarms. Astrée reports 1 743 code locations with alarms from the following categories:

Alarm Category	#Loc
Division or modulo by zero	4
Uninitialized variables	27
Invalid usage of pointers and arrays	310
Invalid ranges and overflows	1 402

With reduced precision settings the analysis reaches 46% of the code, analysis time is 3h7min, the required memory consumption is 5.4GB. The reason of the low percentage of reached code is incomplete environment information, and also the lack of some parts of the application which have not been available to us.

In total the number of alarms about invalid concurrent behavior is 5 759:

Invalid Concurrent Behavior	#Loc
Read/write data race	3 152
Write/write data race	2 599
Invalid usage of OS service	8

Also this project uses enable/disable interrupt calls as a synchronization mechanism and exploits task priorities to implement lightweight synchronization. When support for these mechanisms is finished we expect the number of data race alarms to be significantly reduced.

6 Related work

Applying formal methods to the verification of concurrent programs and systems has a long history. We will focus on recent work and refer the reader to [31] for a survey and historical perspective. The theoretical foundation of Astrée is based on the abstract interpretation theory [7]. We refer the reader to [8] for an in-depth comparison of abstract interpretation techniques with other formal methods. Other tools based on abstract interpretation include Polyspace [9] which can detect shared variables and take task interleavings into account. However, to the extent of our knowledge, it does not report data races nor lock/unlock defects and lacks a direct support for OSEK applications so that users have to manually specify the concurrency setup. By comparison, in addition to reporting all potential data races and lock/unlock defects Astrée provides a complete and automated support for OSEK, including a stub OS library, a toolchain allowing the analysis to be automatically configured by an OIL file, the automatic detection of the entry points of all the tasks and interrupts, as well as the detection of critical sections. The modeling of concurrent embedded operating systems for use in the analysis of applications has been considered before in [11]. We report in [25] the use of Astrée for avionics application and detail the modeling of the ARINC 653 OS specification.

The thread-modular semantics employed to achieve a scalable analysis of concurrent programs is inspired from the rely-guarantee principle, introduced in proof methods [18]. Note that, unlike proof-based verification tools, Astrée automatically infers memory invariants as well as interferences and does not rely on the programmer to provide them.

Another popular method to verify concurrent systems is model checking. Model checking can suffer from the state explosion problem, particularly acute when considering concurrent systems. It has been partially addressed by partial order reduction methods [12]. In practice, the SPIN model checker has been used [13] to check for data-races and deadlocks in concurrent code from NASA. The analysis of C code was however limited to fragments of a few hundred lines. The study also mentions that C code up to 45 KLoc could be handled by analyzing a hand-crafted 1 Kloc model. By contrast, Astrée scales to million-line codes. It does not require building a model by hand, and can analyze directly full C applications without the need to extract small, self-contained parts, which is time-consuming and error-prone. Another recent proposal to improve the scalability of model-checking is to analyze a system only up to a fixed, generally small number of context switches [28]. While this method can be useful to find bugs, it is unsound and only covers a small fraction of the possible behaviors, and is thus not adequate according to the most stringent certification processes used in embedded critical software (such as avionics software [30]). By contrast, Astrée is sound and will find all run-time errors and data-races.

Sequentialization [29] suggests a reduction from concurrent programs to equivalent sequential ones in order to apply existing sequentialization verification methods. The method has been applied in particular to the static analysis by abstract interpretation of interrupt-driven programs [33]. The method is however limited to specific scheduling policies, as a higher priority task must complete before the control is returned to a lower priority task. Unlike Astrée, it does not permit arbitrary preemption (as found for instance in ARINC or POSIX threads), and is thus less general.

With the rise of multi-core applications, formal methods have been updated to take into account weakly memory models. Astrée is also aware of weakly memory models, through a careful selection of which abstractions are employed during the analysis. Similar results concerning the influence of the abstraction on the soundness in weak memory models can be found in [5].

Future work. Future work on Astrée is planned to address its current limitation. Firstly, we plan to add a deadlock detector. Secondly, we plan to improve the handling of thread priorities, including dynamic priorities and the priority ceiling protocol implemented in OSEK, to improve the precision. We plan to add a precise, flow-sensitive tracking of interrupt enable, which will also improve the precision by removing spurious interferences from disabled interrupts. Thirdly, we wish to improve our support for multi-core applications since our current support for multi-core requires, for soundness, ignores the priority of threads and assuming arbitrary preemption. The focus on multi-core will also encourage us to seek more precise abstractions of weakly consistent memory models.

7 Conclusion

Safety requirements mandate that critical software is exempt from run-time errors. The rising predominance of concurrent software architectures puts a strain on classic validation methods, such as testing or code reviews, that hardly cope with the non-deterministic nature of concurrent programs, the huge number of interleavings, and the difficulty to uncover errors in extremely rare but possible cases. We have presented Astrée, a tried static analysis verification tool based on abstract interpretation, and its recent extension to the sound analysis of concurrent C programs, efficiently covering all possible interleavings and uncovering all run-time errors and data races. We have explained how Astrée can support programs for various operating systems and concurrency libraries (POSIX threads, ARINC 653, OSEK/AUTOSAR) and presented encouraging experimental results. Ongoing work includes further experimentation (in particular on automotive applications under the OSEK/AUTOSAR system), support for more systems and concurrency models, as well as the design of additional abstractions to improve both the precision and the scalability of the analysis.

Acknowledgement. The work presented in this article has been supported by the German BMBF project FORTISSIMO and the project ANR-11-INSE-014 from the French Agence nationale de la recherche.

References

[1] AUTOSAR (AUTomotive Open System ARchitecture). http://www.autosar.org.

[2] nxtOSEK/JSP. http://lejos-osek.sourceforge.net/.

[3] AbsInt. *The Static Analyzer Astrée– User Documentation for AAL Annotations*, 2015.

[4] Aeronautical Radio Inc. ARINC 653. http://www.arinc.com.

[5] J. Alglave, D. Kroening, J. Lugton, V. Nimal, and M. Tautschnig. Soundness of data flow analyses for weak memory models. In *Proc. of the 9th Asian Symp. on Programming Languages and Systems (APLAS'2011)*, volume 7078 of *LNCS*, pages 272–288, Dec. 2011.

[6] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A Static Analyzer for Large Safety-Critical Software. In *Proc. of PLDI'03*, pages 196–207. ACM Press, June 7–14 2003.

[7] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'77*, pages 238–252. ACM Press, 1977.

[8] P. Cousot, R. Cousot, J. Feret, A. Miné, L. Mauborgne, D. Monniaux, and X. Rival. Varieties of Static Analyzers: A Comparison with ASTRÉE. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007*, pages 3–20. IEEE Computer Society, 2007.

[9] A. Deutsch. Static Verification of Dynamic Properties. *ACM SIGAda 2003 Conference*, 2003.

[10] J. Feret. Static analysis of digital filters. In *Proc. of ESOP'04*, volume 2986 of *LNCS*, pages 33–48. Springer, 2004.

[11] A. Gamatié and T. Gautier. Synchronous modeling of avionics applications using the SIGNAL language. In *Proc. of the 9th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS'03)*, pages 144–151. IEEE Computer Society, 2003.

[12] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*. PhD thesis, University of Liege, Computer Science Department, 1994.

[13] G. J. Holzmann. Mars code. *Commun. ACM*, 57(2):64–73, Feb. 2014.

[14] ICube. NXT CESAR project. http://icube-avr.unistra.fr/en/index.php/NXT_CESAR.

[15] IEEE Computer Society. Standard for binary floating-point arithmetic. Technical report, ANSI/IEEE Std. 745-1985, 1985.

[16] IEEE Computer Society and The Open Group. Portable operating system interface (POSIX) – Application program interface (API) amendment 2: Threads extension (C language). Technical report, ANSI/IEEE Std. 1003.1c-1995, 1995.

[17] ISO/IEC JTC1/SC22/WG14 working group. C standard. Technical Report 1124, ISO & IEC, 2007.

[18] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, Jun. 1981.

[19] D. Kästner and J. Pohland. Program Analysis on Evolving Software. In M. Roy, editor, *CARS 2015 - Critical Automotive applications: Robustness & Safety*, Paris, France, Sept. 2015.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558–565, July 1978.

[21] A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *Proc. of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'06)*, pages 54–63. ACM, Jun. 2006.

[22] A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[23] A. Miné. Static analysis of run-time errors in embedded real-time parallel C programs. *Logical Methods in Computer Science (LMCS)*, 8(26):63, Mar. 2012.

[24] A. Miné. Relational thread-modular static value analysis by abstract interpretation. In *Proc. of VMCAI'14*, volume 8318 of *LNCS*, pages 39–58. Springer, Jan. 2014.

[25] A. Miné and D. Delmas. Towards an Industrial Use of Sound Static Analysis for the Verification of Concurrent Embedded Avionics Software. In *Proc. of the 15th International Conference on Embedded Software (EMSOFT'15)*, pages 65–74. IEEE CS Press, Oct. 2015.

[26] MISRA-C:2004 Guidelines for the use of the C language in critical systems, Oct. 2004.

[27] MISRA-C:2012 Guidelines for the use of the C language in critical systems, Mar. 2013.

[28] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *Proc. of the 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

[29] S. Qadeer and D. Wu. KISS: Keep it simple and sequential. In *Proc. of the ACM SIGPLAN Conf. on Programming Languages Design and Implementation (PLDI'04)*, pages 14–24. ACM, June 2004.

[30] Radio Technical Commission for Aeronautics. RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification, 2011.

[31] M. C. Rinard. Analysis of multithreaded programs. In *Proc. of the 8th Int. Symp. on Static Analysis (SAS'01)*, volume 2126 of *LNCS*, pages 1–19. Springer, Jul 2001.

[32] P. Sewell, S. Sarkar, S. Owens, F. Zappa Nardelli, and M. Myreen. x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors. *Comm. ACM*, 53, 2010.

[33] W. Wu, L. Chen, A. Miné, D. Dong, and J. Wang. Numerical Static Analysis of Interrupt-Driven Programs via Sequentialization. In *Proc. of the 15th International Conference on Embedded Software (EMSOFT'15)*, pages 55–64. IEEE CS Press, Oct. 2015.

Bringing SPARK to C developers

Authors: Johannes Kanig (AdaCore), Quentin Ochem (AdaCore), Cyrille Comar (AdaCore)

Abstract

Selecting a language in a safety critical application is often a choice dictated by constraints beyond bare technical merits. Availability of tools or internal resources at the time of decision is often critical. However, once such a choice is made, it is extremely difficult to revert. This is very visible in domain such as avionics or automotive where code bases are sometimes created and maintained over decades. Rewriting software is just not an option.
As such, many software teams live with technical choices that can't be questioned, or marginally. This is notably the case in the world of the C programming language. Its defects are well documented and have been known for many years. An entire sector of the tool industry is focused on developing workarounds in the form of code analyzers, coding standards or auto-testing tools. Other languages and environments are known to provide results at lower cost. However, the barrier of entry, software re-writing, is often beyond what is industrially acceptable.
In this paper, we will discuss one of these alternatives, the SPARK language. We will describe a framework that allows to gain direct benefits from early investment phases and we will discuss supporting tools currently under development.

1. Introduction

Programming languages are usually designed in terms of how well they can express executable semantics and software architecture. They are seldom designed to optimize behavior regarding specification and verification. This explains how difficult it is for a static analysis tool to analyze a piece of C code without either reporting a lot of false alarms or missing a lot of problems. More modern languages such as Ada tend to exhibit better results – although still not entirely satisfactory.
The SPARK language [1] was designed the other way around – with static verification in mind from the beginning. In order to avoid reinventing an entire development environment, it was decided to use the syntax and the executable semantics of an existing language not too far off the overall goal. In this case, the Ada 2012 language. However, all functionalities that were not congruent with static analysis objectives have been removed. This includes in

particular so called access types (pointers) and exception handlers. To this foundation, static semantics have been superposed to some of the existing language notations, notably contracts and assertions. A number of additional annotations (pragmas and aspects) came in to complete the picture.
With the above in mind, a SPARK program can take advantage of state-of-the-art proving technologies (The Why3 program verification platform [2], SMT-solvers such as CVC4 [3], Altergo [4] and Z3 [5], manual provers such as Isabelle [6] and Coq [7]) and be formally verified regarding various aspects such as data coupling, absence of run-time errors or functional correctness. When formal methods are not usable, dynamic verification can still be used to verify most of the SPARK annotations, through regular testing and assertion checking.
Taking full advantage of SPARK however requires making the choice of designing the software with that language very early on. There are various examples in the recent history of teams that have successfully done so, but such opportunities of rewrite are merely an exception. In practice, most developments are done in full Ada or C, and this cannot be changed after years of development, because of the high cost of rewriting existing code. So many projects are stuck with a less-than-ideal choice made at the very beginning of the project, which can’t be changed.
Another problem is that, in projects where a custom processor is used, C is often the only language for which a compiler is provided.
There is a solution. In this paper, we describe two ways to make a project adopt SPARK in small steps, getting benefits for all investments. We also propose a solution for the case where only a C-compiler is available.

2. Current Situation of Many Projects in the Industry

The default programming language used in the high integrity embedded industry today is and remains the C language. There are many reasons for that - C is a relatively simple language, the resulting code is fairly efficient, there is usually no abstraction layer between the code generated from the compiler and the execution hardware, many libraries and off-the-shelf components are available, many trained developers can be found on the market place, tools are fairly comprehensive and supported on almost every single hardware platform. In many situations, it’s also the default choice, that is a choice that does not get questioned by management.
Unfortunately, C is also a language with a large list of very well documented flaws and vulnerabilities [8]. Various tools and techniques have been developed as an attempt to workaround and get this flaws under control, from language subsets (such as MISRA) to a wealth of static analysis and testing tools. The need to master these tools and techniques however reduce some of the benefits mentioned before (in particular the availability of trained developers).
The predominance of C in the current industry is such that it is not reasonable to expect a shift in practice. The size and longevity of existing components is such that rewriting them is not economically feasible. C is and will remain a strong actor no matter what. Interestingly, the same is true for Ada. Many systems developed in the 80’ and 90’ were done in Ada, the

default language for military applications at the time, since it was mandated by the DOD. Although the attractiveness of the technology temporarily decreased at the turn of the millennium, the user base in the A&D domain stayed strong and consistent. This is probably thanks to the same mechanics and trends that it is possible today to find not only a strong Ada toolset and environment, but means to take advantage of it even in the context of a strongly C-based culture.

Another trend should also be emphasised: multi-language programming. It's now extremely common to see projects developed using a multitude of different languages. In a single executable, it's not rare to find C linked with C++ or Ada, sometimes interacting with virtual machines running Java, Python or C#. To a software architect, the preexistence of an appropriate component often matters more than the language in which said component is initially developped. Not to mention migration paths, where developers want to migrate from X to Y technology, while keeping legacy components developed in X. In effect, it's common to only migrate new components while still taking advantage of years of previous development.

To summarize, if a different language such SPARK should be adopted, it must be able to interoperate with existing languages, and in particular with the C language.

3. Short Presentation of SPARK

SPARK is more than a simple programming language - it is also a specification language and a verification system. We will explain these aspects here, as they are essential to the understanding of the remainder of the paper.

3.1. SPARK - the Programming Language

SPARK is a programming language that has been designed from the beginning with safety, maintainability and verifiability in mind. The new starting point was the Ada programming language, from which a number of features have been removed which are considered dangerous in safety-critical programming. The most important features that have been removed are pointers and exception handling. However, a recent major redesign of the language has been based on Ada 2012 [9], the latest version of the Ada language, which contains many features that are particularly useful for safety-critiical programming, such as contracts.
Ada is a very rich language and SPARK inherits many of the powerful features of Ada. The rich type system allows to specify specify ranges for integer and floating-point variables, which is much more precise than simply using the predefined word sizes. Array types are much safer and more powerful. In particular, they do not require the use of pointers, and the length of an array can be queried from it. Common array operations such as initialization, copying, slicing and concatenation are built-in.

3.2. SPARK - the Specification Language

However, SPARK is more than a programming language, it also includes a powerful and expressive specification language, which allows to specify the behavior of the program. The most common specifications are attached to functions and are able to describe the behavior of the function in great detail:

- The *Global* specification describes which global variables are accessed by this function, and whether they are read, written, or both. The same property for parameters is already covered by regular Ada syntax.
- *Pre- and Postcondition* (which already exist in Ada 2012) are boolean expressions that must be true at the beginning and the end of the function, respectively. Together they express what the function *requires* to work (the precondition) and what it *guarantees* when returning.

At a larger scale than a single function, other specification features exist, such as abstract state, which allows to group together the state of some package into a single logical variable, and elaboration/initialization properties, which allow to state the properties of data that is set-up only once at the beginning of the program.

It should be noted that all specification features are completely optional.

3.3. SPARK - The Verification System

Ada (and SPARK) allows very easy checking of safety properties such as division by zero, arithmetic overflow and buffer overflow, simply by providing run-time checks. For example, for division by zero, before carrying out the division, it is checked that the divisor is different from zero. If not, the program stops and an error is reported. This feature can be enabled or disabled using a compiler switch, and it is already very useful to find silly errors during testing.

In a similar manner, most of the specification features of SPARK can be checked when the program is run, e.g., during testing. For example, pre- and postconditions are handled like assertions; with a simple compiler switch, they can be compiled into the binary and will be checked during the run of the program.

In addition to this, the SPARK language comes with verification tools that take a SPARK program with specifications and check it for errors statically (that is, without running the program). There are a number of different individual verifications done by the tool. It checks that:

- All variables are properly initialized before they are used;
- Every function only reads and writes the specified global variables and parameters;
- No so-called run-time errors such as division by zero, arithmetic overflow or buffer overflow can occur;

- When an function is called, its precondition is true, so that the function is called in a valid state according to its specification;
- When a function returns, its postcondition is true so that the function returns in a valid state according to its specification.

When applied full-scale, the verification of SPARK can guarantee that no errors of the above kind appear in the program.

The SPARK toolset is based internally on the Why3 verification platform, and various SMT solvers such as CVC4, Alt-Ergo and Z3.

The remainder of this paper will argue that SPARK is useful even when not used everywhere with full verification, but only one or two aspects of SPARK are integrated into the development process.

4. A mixed SPARK-and-C environment

The SPARK language is designed to perform unit verification. In other words, it proves correctness of a subprogram (function) according to its own specification and the one of its dependencies (callees). One can prove a subprogram assuming its precondition, verify that the precondition of its dependencies, assuming that the postcondition of its dependencies holds, and finally proving the postcondition. There are two strong assumptions that need to be verified – the subprogram precondition and the postconditions of the dependencies. In a perfect world, all of those are also verified through formal methods, but this is not a requirement of the language. Traditional testing can also be used as an alternate means of verification.
From a practical point of view, SPARK declarations (ideally with contracts, but this is not strictly required) for all called functions are all that is needed for the SPARK tools to carry out full verification of a given SPARK function. For the called functions, other implementation languages can be chosen, such as C. The SPARK verification results are correct, provided that the other functions correspond to their specification.
Several different use cases for this technique are possible. In many projects, legacy C code exists and it would be undesirable to rewrite it; but new developments should happen in some stricter and safer environment such as SPARK. Often, device drivers for some component of the system are written in C. Or maybe only the most safety-critical component of a system shall be written in SPARK, and other components implemented in C. Another approach could be to progressively migrate the logic of the application in SPARK while keeping standard components such as drivers, libraries and OS written in C. For any such use case, using the above technique, it is possible to use SPARK as the implementation language only for the components where this is desired. The full SPARK specification and verification features can still be used on this part of the project.

```
typedef struct {
  void** data;
  int pointer;
  int len;
  int capacity;
} t_queue, *pqueue;

pqueue init_queue(int capacity);

void* queue_pop(pqueue q);

void queue_enqueue (pqueue q, void* elt);

bool queue_is_empty(pqueue q);

int queue_length(pqueue q);

void free_queue(pqueue q);
```

Fig. 1: The C interface of a simple queue implemented using a ring buffer.

Of course, the verification of the component(s) written in SPARK is only valid if the assumptions on the other components, which have been used to complete the SPARK verification, indeed implement their contract [10]. As the SPARK tools can only be applied to SPARK programs, other verification methods must be applied. Thanks to executable contracts, at least the contracts specified at the boundary between the SPARK and C code can be very easily verified using testing.
Overall, this approach provides a very pragmatic path to migration with quick return on investment. As soon as one subprogram is converted, migrated and proven correct, the overall safety of the application gets improved.

5. SPARK as a language for C specification

But the SPARK language can be useful even when the verification features and the programming language are not used. Verification is always done according to a specification, and the C language is very weak at expressing specifications. As such, and as we have shown in the previous section, the SPARK language has a specification semantics much richer than most programming languages, in particular compared to C. This includes some well-known Ada features (strong typing, parameter modes…) Ada 2012 additions (pre/post conditions, quantifiers…) and SPARK complements (data coupling, states…).
So one may want to use the SPARK specification language to specify C programs. This sounds like a surprising idea at first, but it is entirely possible and useful, as we will show in this section. We will also show an example of this use case at work.

```
type data_arr is array (int range <>) of Address;
subtype data_arr_constr is data_arr (int range 0 .. int'last);
type data_arr_access is access data_arr_constr;

type t_queue is record
   data : data_arr_access;
   pointer : aliased int;
   len : aliased int;
   capacity : aliased int;
end record;

type pqueue is access all t_queue;

function Is_Empty (Q : pqueue) return Boolean is (Q.len = 0);

function To_Arr (Q : pqueue) return data_arr is
 (if Q.pointer + Q.len <= Q.capacity then
   Q.data (Q.pointer .. Q.pointer + Q.len - 1)
  else
   Q.data (Q.pointer .. Q.capacity - 1) &
   Q.data (1 .. Q.len - (Q.capacity - Q.pointer)));

function Pop (A : data_arr) return data_arr is (A (A'First + 1 ..  A'Last));

function First (A : data_arr) return Address is (A (A'First));

function init_queue (cap : int) return pqueue with
  Post =>
   Is_Empty (init_queue'Result) and init_queue'Result.Capacity = cap;

function queue_pop (Q : pqueue) return Address with
  Pre  => not Is_Empty (Q),
  Post => Q.len = Q.len'old - 1 and
          Pop (To_Arr (Q)'Old) = To_Arr (Q) and
          queue_pop'result = First (To_Arr (Q)'Old);
```

Fig. 2: The corresponding SPARK interface with contracts.

The key idea is add a SPARK wrapper to a C function, with identical parameter profile, and redirect all calls to this C function in the program to call the wrapper instead. If the wrapper has SPARK contracts such as Pre- and Postconditions, these contracts get dynamically checked on every call. If the contracts are complete (i.e. capture most if not all of the requirements), this provides a great way to check the requirements of the software during testing in a very explicit yet convenient way.
In fact, much of the mechanical part of this outlined procedure can be automated: the --fdump-ada-spec option of gcc [11] can be used to generate a SPARK wrapper function with an identical parameter list, and the usage of Import and Export pragmas can achieve the rerouting of the calls. Compiling and linking everything together, and enabling assertions when the SPARK-part is compiled, will achieve the desired contract checking.
The idea here is to use C as the language for implementation, and SPARK as the language for specification. Said otherwise, in this mode we only use the “specification language” aspect of SPARK.

Of course, in this case, it is not possible to formally verify a C implementation against a SPARK specification, although the integration with some existing C verification techniques relying on similar technologies (integrated with Why3) could be envisioned at some point. What we are focussing on here is the executable semantics of SPARK specification, which can be checked at run-time. In other words, in this mode, it is possible to compile a hybrid C/SPARK application, and activate specification verification during execution.
This can have various benefits. One is during unit tests, to identify inconsistent calls or invariant breaches early on. It is also a great way to strengthen stubbing and verify that stubs are called with the same constraints as actual code, thus reducing the number of software to software integration errors.
If carefully placed, it can also be kept in final deployment, as a way to protect a component from misuse. The executable specification then acts as a barrier to users, making sure that invariants and assumptions are respected at run time. That is also a nice place to exhibit all the verification code that would otherwise end up being developed as defensive code. One direct application of that can be to contribute to the argumentation of Freedom From Interference [12] as required by ISO-26262 [13].
We now proceed to the promised example. It stems from a small C program which is part of the Why3 platform. The code implements a simple queue with enqueue and pop functions (see Fig. 1). The corresponding SPARK interface (see Fig. 2) has been generated with the `--fdump-ada-spec` switch of gcc, and then manually modified later to improve the mapping. For example we us a few tricks to use SPARK arrays instead of access types to represent the queue data. The SPARK contract uses a “model function” called `To_Arr` to map the queue structure, which wraps around the array in the implementation, to a plain array with elements in the right order for specification purposes. The contracts can are always expressed on that plain array instead of the complex actual data structure, and can now be expressed using straightforward SPARK array operations such as concatenation and slices. For example, the `queue_pop` function has a postcondition which states that the queue after the pop (simply named `Q`), mapped to the plain array, gives the same result as mapping the queue *before* the pop (using the syntax `Q’Old`) to the plain array, and then removing the first element. Looking carefully at the contract of `queue_pop`, one can see that in fact the complete desired functional behavior has been described using the contract.
The reader might ask if the same cannot be achieved using simple assertions at the beginning and end of the C function. But as the example below shows, such a complete contract is difficult to express in C because of the lack of high-level language features such as array concatenation and slices.
The `enqueue` function has not been specified, but this is not a problem, Using this approach, one can selectively apply it to only the desired functionality of a package.

6. Back to the C

So far, we’ve assumed that the final application was a mixed of SPARK and C files compiled and linked together. This requires the availability of both an Ada 2012 compiler and a compatible C compiler. While the number of platforms supporting Ada 2012 is quite large, it is often the case that there is no compiler available for more very specific or custom targets.

The absence of such technologies may render the whole discussion above futile for projects targeting such platforms. However, virtually all platforms come with a C-compiler.
There is an alternative to this, which is to consider the C language as an intermediate representation in the SPARK code generation chain. In other words, compile SPARK to C with a “standard” toolchain, and then C to assembly with the specific target compiler, which acts as a back-end in this context. It is important to realize that generated C code is not intended to be readable or modifiable. Although MISRA properties can be enforced during code generation as to ensure optimal portability and safety of the generated C layer, the SPARK and C programming languages are too far to generate C that would look like something written by a human being. In particular, the SPARK to C generator may go through expansion or optimization phases, or make choices to translate high-level SPARK concepts into low level C concepts in a way that is not efficiently manageable by a developer. But that is not a problem: just as one would not modify the assembly generated from a compiler except in very specific cases, one should not worry about modifying this C code – which in effect acts here as a universal assembly language.
Going this route is more than just generating C. Of course, all the benefits of SPARK, such as strong typing, runtime checks, static analysis and formal verification are still present.
Having a regular Ada cross compiler has still significant advantages over the SPARK to C generation path. Among other things, it simplifies the integration with tools such as debuggers and makes tool vendor validation easier. However, using the C language as an intermediate language here allows to virtually provide a universal SPARK compiler.

7. DO-178C certification considerations

The various tools and technologies presented so far still need to be properly articulated around DO-178C to be usable in certified avionics context. This implies both references to the certification objectives that can be targeted and corresponding qualification or certification material.
Looking at the SPARK language, its contributions to certification credits start at the planning phase on DO-178, targeting activities such as 4.4.1.a “software development should be chosen to reduce its potential risk to the software being developed”. SPARK is exempt of a number of programming vulnerabilities without the need of any additional coding standard or tooling. When looking at other potential credits, 6.3.4.f comes to mind - “accuracy and consistency”. SPARK can help demonstrating absence of a number of problems identified there, such as fixed point arithmetic overflow and resolution, floating point arithmetic, used of uninitialized variable and unused variables. To achieve this specific objectives, the verification toolchain associated with SPARK needs of course to be qualified TQL-5.
If data flow is specified in SPARK, the verification of its implementation as requested by 6.3.4.b can also be automatically verified by the TQL-5 qualified verification chain.
Going one step further, SPARK can be taken advantage of following methodologies experimented by Airbus [14] and described in the formal proof supplement DO-333. In this case, formal proof can be used to replace some or all of the low level testing. This is assuming both a higher level of tool qualification (TQL-4) together with an argument

demonstrating that the object code is indeed correctly translated from the source code (so called "preservation of properties").
Regarding the last technology presented, the SPARK-to-C compiler, it's important once again to realize that in this configuration, the C language is merely an intermediate step in the compilation process. In other words, the generated C code should not be considered as source code by DO-178 definitions, but rather as an intermediate representation within a toolchain. As a result, the source code is SPARK, and the aggregated {SPARK-to-C, C-target} compiler should follow the same recommendations and objectives as other compiler. For example, element to stored in version control is the SPARK source code, structural code coverage has to be achieved at the SPARK level, a source to object traceability study has to be performed at level A, etc.

8. Conclusion

We've demonstrated in this paper that the entry barrier to technologies such as SPARK can be lowered significantly. SPARK interoperates well with C programs, so that usage of the SPARK language can easily be limited to new code or particularly critical code at first. Or one can add SPARK contracts to C code without any SPARK implementation, and still benefit from a powerful language to express requirements and runtime checking. Finally, thanks to the SPARK-to-C technology, SPARK can be applied even in a context where no Ada-2012-Compiler is available for the selected target.
Technologies to support these methodologies are being developed. Others tools could have been mentioned as well and be described in other papers – notably the role of C or SPARK code generation from modeling languages such as Simulink. Overall, these tools and methodologies demonstrate not only the feasibility but the low risk associated to an iterative path of experiment and deployment.

References

[1] SPARK, http://www.spark-2014.org
[2] Why3 verification platform, http://why3.lri.fr
[3] CVC4, http://cvc4.cs.nyu.edu/
[4] Alt-ergo, http://alt-ergo.ocamlpro.com/
[5] Z3, https://z3.codeplex.com/
[6] Isabelle, https://isabelle.in.tum.de/
[7] Coq, https://coq.inria.fr/
[8] Paul E. Black, Michael Kass, Michael Koo & Elizabeth Fong, *Source Code Security Analysis Tool Functional Specification*, NIST, 2011)
[9] Ada 2012 Reference manual, http://www.ada-auth.org/standards/ada12.html
[10] Johannes Kanig, Rod Chapman, Cyrille Comar, Jérôme Guitton, Yannick Moy, Emyr Rees: Explicit Assumptions - A Prenup for Marrying Static and Dynamic Program Verification. TAP 2014: 142-157

[11] Emmanuel Briot - AdaCore Gem 59: Generating Ada Bindings for C Headers, http://www.adacore.com/adaanswers/gems/gem-59/
[12] Handbook for Functional Safety, Software Partitioning Edition, JASPAR H-FSS-07-0002, 2013
[13] Road Vehicles - Functional Safety, ISO 26262-1, 2010
[14] Frank Dordowsky, *An experimental Study using ACSL and Frama-C to formulate and verify Low-Level Requirements from a DO-178C compliant Avionics Project*

Spreading Static Analysis with Frama-C in Industrial Contexts

A. Stéphane Duprat[1], B. Victoria Moya Lamiel[1],
C. Florent Kirchner[2], D. Loïc Correnson[2],
E. David Delmas[3]

1: Atos, 6 Impasse Alice Guy, B.P. 43045, 31024 Toulouse Cedex 03
2: CEA LIST, Software Safety Laboratory, Saclay, F-91191
3: Airbus Operations S.A.S., 316 route de Bayonne, 31060 Toulouse Cedex 9

Abstract: This article deals with the usage of Frama-C to detect runtime-errors. As static analysis for runtime-error detection is not a novelty, we will present significant new usages in industrial contexts, which represent a change in the ways this kind of tool is employed.

The main goal is to have a scalable methodology for using static analysis through the development process and by a development team.

This goal is achieved by performing analysis on partial pieces of code, by using the ACSL language for interface definitions, by choosing a bottom-up strategy to process the code, and by enabling a well-balanced definition of actors and skills.

The methodology, designed during the research project U3CAT, has been applied in industrial contexts with good results as for the quality of verifications and for the performance in the industrial process.

Keywords: Static analysis, abstract interpretation, safety critical software, embedded system, modular analysis, Frama-C, ACSL, runtime error

1. Introduction

Static analysis for runtime-error detection is not totally new; different tools have been proposed since fifteen years. Nevertheless, it is not a widespread practice even in critical software. Static analysis is commonly employed by specialists for independent verifications and after the development of the program. This activity is a good way of improving quality but it is often synonym of additional activity in the main process and additional costs.

In order to facilitate the usage of static analysis, we conducted during the research project U3CAT, methodological studies and tooling development. The main objectives were: good coverage of runtime-errors, scalability, predictable costs, and a good integration in the development cycle.

We largely succeeded in this endeavour: this article reports on the main difficulties encountered, the technical and methodological solutions adopted, and the benefits obtained.

The produced methodology has been updated and used to answer to specific needs in industrial contexts and we report this industrial experience.

Finally, we'll conclude on new usages already identified, but not yet used in industrial context.

2. Context

The Frama-C source code analysis platform

Frama-C is an Open-Source platform dedicated to the analysis of C programs. It differs from other code analysers as it provides a diverse set of formal tools, cooperating through code annotations written in the ACSL language. ACSL is a behavioural specification language that can express a wide range of functional properties, through partial or complete specifications. Analysers themselves may report results in terms of new ACSL properties asserted inside the source code.

Frama-C [6] is built around a kernel that performs the parsing and type-checking of C code and accompanying ACSL [5] annotations if any, and maintains the state of the current analysis project. This includes in particular registering the validity status of all ACSL annotations. Analyses themselves are performed by various plugins that can validate annotations, but also emit hypotheses that may eventually be discharged by other plugins. This mechanism allows some form of collaboration between the various analysers.

Two important analysis plugins are Value Analysis [7] and WP [8]. Value analysis is based on abstract interpretation, and computes an over-approximation of the values that each memory location can take at each program point. When evaluating an expression, Value Analysis will check whether the abstraction obtained for the operand represents any value that would lead to a runtime error[1]. For instance, when dereferencing a pointer,

[1] Rutime errors include: division by 0, undefined logical shift, overflow, underflows on integers, use of non-initialized variable, dangling pointer, invalid memory access , use of non-allocated pointers, problem of overlapping lvalue assignment, undefined side-effect in expressions, and invalid function pointer access.

the corresponding abstract set of location should not include NULL. If this is the case, Value Analysis emits an alarm, and attempts to reduce the abstract value. In our example, it will thus remove NULL from the remaining abstract state. The analysis is correct, in the sense that if no alarm is emitted, no runtime error can occur in a concrete execution. It is however incomplete, in the sense that some alarms might be due to the over-approximations that have been done and might not correspond to any concrete execution. Various settings can be selected to choose the appropriate trade-off between the precision and the cost of the analysis. While the most immediate use for Value Analysis is to check for the absence of runtime error, it will also attempt to evaluate any ACSL annotation it encounters during an abstract run. Such verification is however inherently limited to properties that fit within the abstract values manipulated by Value Analysis. Mainly, it is possible to check for assertions on bounds of variables at particular program points.

WP is a deductive verification-based plugin. Contrary to Value Analysis, which performs a complete abstract execution from the given entry point, WP operates function by function, on a more modular basis. However, this requires that all functions of interest as well as their callees be given an appropriate ACSL contract. Similarly, all loops must have corresponding loop invariants. When this annotation work has been completed, WP can take a function contract and the corresponding implementation to generate a set of proof obligations – logic formulas whose validity entails the correction of the implementation with respect to the contract. WP then simplifies these formulas, and sends them to external automated theorem provers or interactive proof assistants to complete the verification. WP's main task is thus to verify functional properties of programs, expressed as ACSL annotations. It is however also possible to use it to check that the pre-conditions written for a given function f imply that no runtime error can occur during the execution of f.

Frama-C is already used in this industrial context. First usage at Airbus is for an implementation of a coding rule checker called Taster [3] and a second one, Fan-C [4], targets verification of data and control flow based on semantic analysis.

Main principles of a static analysis project

One of the main principle of Value Analysis-based projects is that to computes values of variables for all possible program execution, either starting from the program's 'main' function or another one expressed to the tool by an option on the command line.

The Value Analysis handles the semantics of the C program, but not only. One strength of the tool is to be able to perform analyses on incomplete programs, that is, pieces of source code not containing all definitions of functions called.

The user can define function contract in ACSL defining behaviour of the function and the tool is able to integrate, within its analysis, the semantic of the C program and the semantic of the ACSL.

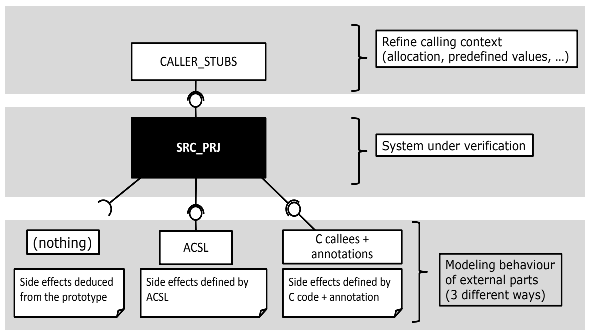

CALLER_STUBS
Refine calling context (allocation, predefined values, …)
SRC_PRJ
System under verification
(nothing)
ACSL
C callees + annotations
Side effects deduced from the prototype
Side effects defined by ACSL
Side effects defined by C code + annotation
Modeling behaviour of external parts (3 different ways)

Figure 1 : Topology of a static analysis project

For a C function without C definition neither ACSL contract, the tool is able to consider a default behaviour deduced from its prototype.

Finally, the user has three solutions for an external function : (1) nothing, (2) an ACSL contract, (3) a callee stub written in C language and which could use specific Frama-C builtin functions (see Figure 1 : Topology of a static analysis project).

3. Modular analysis

The modular analysis consists in using Value Analysis on small pieces of programs in a consistent manner.

The ACSL language, by defining the behaviour of software interfaces, facilitates the analyses of independent parts of software. Properties defined in the interfaces can be used in two ways: for verification purpose on one hand and for hypothesis definition on the over hand.

This small and trivial example can illustrate both usages:

```
/*@
  requires \valid(p);
  assigns *p;
  ensures \initialized(p);
  ensures 0<= *p <10;*/
extern void get_index(int* p);

/*@ requires \initialized(&x);
  assigns \nothing;*/
extern void bar(int x);
```

```
int foo(int p[10])
{
  int index ;
  int status ;
  get_index(&index);
  bar(index);
  return p[index];
}
```

Figure 2

The called functions get_index and bar are not defined by their source code, but by partial ACSL contracts.
For get_index function:

- For verification purpose at the calling context:
  - The *requires* clause is able to verify that pointer p is valid (referencing an allocated memory)
- To introduce some hypothesis on the behaviour of the function
  - The *ensures* clauses define some hypothesis on the outputs: at the return of the function, the value pointed by b is initialized and in the range [0;10[.
  - The *assigns* clause specifies side-effects. It can be used to define side effects on other locations than those identified by the function parameters.

For bar function

- For verification purpose at the calling context:
  - The *require* clause check that the value of parameter x is initialized
- To introduce some hypothesis on the behaviour of the function:
  - There is no side-effect defined by the *assigns* clause (pure function)

In this way, the verification of the function foo can be performed on this function alone and under the hypothesis of the correct behaviour of the called functions defined by ACSL contract. Verifications are targeting the behaviour of the function itself and also the calling contexts of the called functions. The same contracts of the called functions will then be used in their proper analysis.
Considering the above example, the verification of “foo” doesn’t need the source code of its callees (“get_index” and “bar”), as the ACSL contracts for both callees are sufficient for the analysis. Besides the verification reaches all contexts, including the function behaviour and the calling contexts of the callees, these ACSL contracts will be used lately during their corresponding analysis.
In this way, the different functions i.e. the different subsets of software can be developed and analysed independently and consistently thanks to the function contracts in ACSL.

ACSL contracts can be used not only in the detection of RunTime-Errors, but also on more specific objectives. For example, the verification of functional ranges of parameter values can be verified in this way.

While this approach applies to Value Analysis, ACSL is also able to define functional properties that will be verified with WP using deductive proof techniques as stated in §2.

4. The bottom-up strategy

One of the main issues of static analysis tools in general is that they can produce many false alarms and/or take large amounts of time. The amount of effort necessary is thus difficult to evaluate before performing the analysis. These unpredictable costs and technical difficulties can dampen industrial applications and contractual commitment possibilities.

The most trivial technique to analyse a software is to perform analysis of the whole program completed by a definition of the called libraries. This strategy can be a winning one in case of immediate success. But success is not guaranteed and engineers can be stucked by the number of false alarms and computation time.

One alternative is to divide the program in smaller pieces, to add ACSL contracts in the subset interfaces and to conduct a modular analysis. This strategy can be a solution to obtain a good result, but defining ACSL contracts in a reverse engineering work can be a costly activity.

Facing these difficulties, we defined a bottom-up methodology aiming at succeeding in the analysis of a whole class of programs and with predictable costs.

This methodology is based on an exhaustive analysis of each function with a bottom-up progression. The lowest layer is analysed first and all issues are handled in order to make dispense of all warnings. Different actions are possible: correction in case of bug, fine-tuning of analysis parameters of the tool, or addition of ACSL clauses to help the tool in case of inaccuracy. Once the first layer is treated and all the analyses raise no alarm, each iteration will consist in the integration of sources of the upper layer in their analysis. These iterations will end when reaching the top of the program. This progressive approach has been successfully applied in several analyses. Two experienced uses cases are presented in the §5.

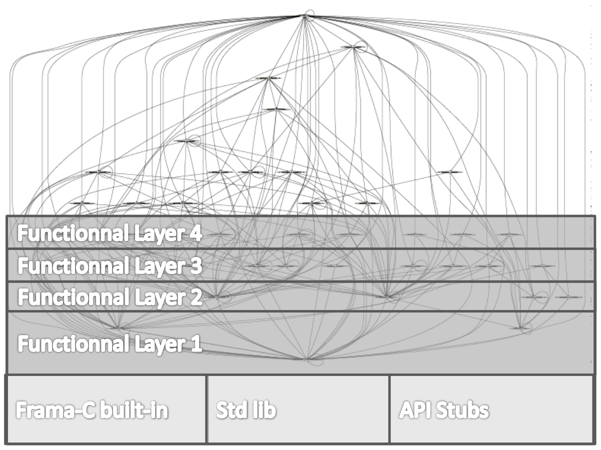


Figure 3: Bottom-Up Strategy applied on 40 C files

By using such a bottom-up strategy, programs that were difficult to analyse as a whole can be analysed step by step up to the main or the entry function.

The opposite strategy of the bottom-up strategy is the top-down strategy consisting in integrating all the source code and performing the analysis on the main function. This strategy can be successful if the whole program is entirely and quickly analysed, otherwise, it leads to the situation previously described and that we encountered where the user is stuck by a large amount of false alarms.

5. Experience report

We report the application of this Bottom-Up strategy on two subset of software. Both are aeronautical software in an intermediate stage of development. Complexity of these two subsets is presented in table below.

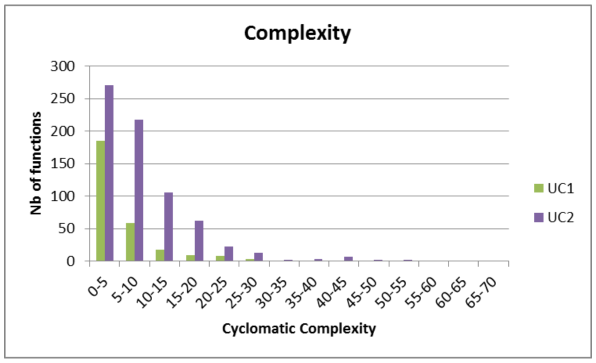


Figure 4

The size of these uses cases are 20 kloc for UC1 and 55 kloc for UC2. UC2 contains complex types considering nested structures, pointers and arrays and with a lot of string manipulations. This higher degree of complexity of UC2 is confirmed by the cyclomatic complexity figures computed by Frama-C. 95% of functions of UC1 have a complexity <15 while 95% of functions of UC1 have a complexity <20.

Objectives

Verification objectives are not expressed in terms of static analysis techniques but by a need of detection of the following threats:

- T1: usage of a non-initialized variable
- T2: usage of a non-initialized pointer
- T3: out of bound access of an array element
- T4: uninitialized output value of a function
- T5: usage of the address of a local value out of the scope of its declaring function
- T6: string management

Solution

A dedicated methodology based on the use of Frama-C's Value Analysis plugin has been proposed to handle all these objectives. Some of these threats are directly handled by usage of the plugin Frama-C/Value. Some additional artefacts have been deployed in order to achieve other objectives.

For example, some callers (as instrumented function on the verification project) have been generated in order to implement the verification of the complete initialisation of all output variables. This mechanism is illustrated for threat T4 in the example below:

```
typedef struct { int a; int b;} S;

void f(S * s1)
{
  s1->a=0 ;
  return ; // filed s1.b is not assigned
}
```

Source code to verify

```
// validation function for f
void caller_f(void)
{
  S l_param1;

  // call of f with parameters to an
uninitialized state
  f(&l_param1);

  // verification that l_param1 i initialized
  //@assert \initialized(&l_param1);
}
```

Caller generated

Figure 5

In this situation, the tool is able de detect that s1 parameter is not fully assigned in the function f.

Application

Following the bottom-up strategy, all functions have been individually analysed, starting from the lower layers.

For each analysed file, the steps are the following

1- Prepare the environment
   a. Define a caller for every analysed function
   b. Define a correct stub for each called function
2- Launch a batch analysis on all functions of the files of the layer
3- Analyse each generated warning
   a. In case of real warning,
      i. Mention the issue on a report file
      ii. Fix the issue
   b. In case of a false warning due to a tool inaccuracy
      i. Help the tool by adding some ACSL annotations in the source code or by using specific options
   c. In case of absence of warning, the analysis of the layer are finished, go the next file
4- Without any warning, the process is finished for this layer, go to the upper layer. Launch a new analysis in case of remaining warnings (go to step 2).

Notes:

- Minor modifications of source code have been made to accelerate the analysis. These modifications concern almost reductions of array size.
- The semantics of parallel execution of the multithread program are not handled

As for the industrial organisation, the analysis has been conducted by a team with the support of an expert. Two persons have been involved in UC1 and three for UC2. Each person has been working on different files. The next files to be analysed were determined with the aid of a “module call graph” consolidated with the list of the already analysed source files and always following a bottom-up progression. All issues reported were checked by another team member and periodical technical meetings were organised.

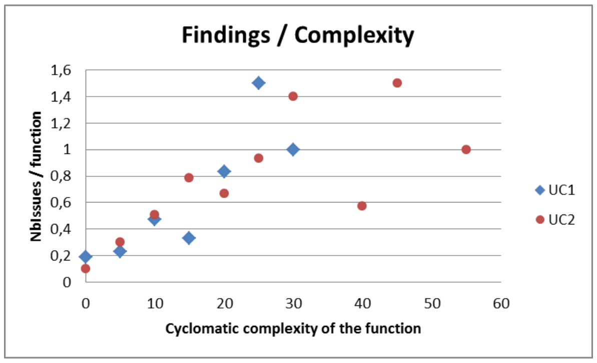
Results

All source files have been analysed. The total number of all findings and their proportions compared to the number of lines of code is presented in the table below.

	UC1	UC2
Findings	54	232
Kloc	17500	57500
Findings / kloc	3,1	4,0

Proportions of findings between both use cases are comparable. Number of warnings in proportion is normally higher for UC2 which is more complex and including string computation.

A closer look reveals a coherent relation in both Use Cases of the medium number of findings per function depending on the cyclomatic complexity of the function. This repartition is quite linear in UC2 for complexity up to 20.



Proportions of the different warning categories are presented in the diagram below.

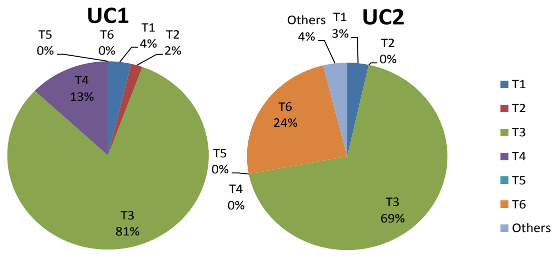


Figure 6

The great majority of findings are concerning array indexes. As the verification is done at a unitary level, these warnings are not meaning that there are as many real bugs detected. It means that indexes of arrays are not checked at each level of functions (this could be at most a lack of robustness, but definitively not necessarily a bug).

Both use cases are diverging on full initialisation of output values: while it represents the second most important category of warnings on UC1, none of them are detected on UC2 thanks to some specific preventive actions conducted on this subset. No warning on string operations are detected on UC1 because of the absence of strings in this subset. One of the most interesting lessons learnt is about the verification of string computation widely used in UC2. String computation is a hard point of verification by static analysis. This experience demonstrates that, under the assumption of some good coding practices (for example: strncpy instead of strcpy) the software can be analysed with enough precision to limit the amount of false alarms.

6. Feedback

Feedback on the Modular analysis strategy

The modular analysis strategy can be a solution for scaling up in the face of major difficulties with full program analysis. But the definition of ACSL contracts of the subset can be a costly activity if it is done in a reengineering process. Another drawback is that some implicit hypotheses that are taken for the analysis of the different modules and not always correct for the whole program. For example, the different locations referenced by pointer parameters of one function are considered as separated locations; that is not always the case. This can lead to an unsound analysis.

On the other hand, if the modular analysis is applied early during development, just after the coding stage and if contracts are already defined as a part of the design, this becomes a good strategy for increasing rapidly software maturity. Each developed subset can benefit from this valued added approach without waiting for the development of all pieces of software. After software integration, the enhanced quality of the produced code will facilitate an analysis of the whole program that will provide the highest level of trust.

A balance can also be made for stub definition between a solution based on ACSL contracts and a solution based on a full C language definition of stubs. The C language offers multiple ways to define a representative behavior of a called function. One advantage is to use only one language for the user and for the tool too. But as presented in §3, the same ACSL contract (ex: `"ensures 0<= *p <10;"`) used as a property of the called function on one hand is also used as a property to verify on the real implementation of the called function. This duality is not allowed by a C definition of a stub.

Feedback on the Bottom-Up strategy

At the provider side (Atos), who proposed the solution and applied it. The bottom-up approach enabled engineers to analyse the entire software cost efficiently and within their deadlines. This success can be explained mainly by the good scalability of this method and by the industrial organisation enabling several actors to work on the same software.

At the side of the industrial customer (Airbus), requester of this analysis, all the verifications requested have been reached and with a very high degree of confidence due to the application of a static analysis by abstract interpretation solution.
For UC2, a large part of findings are concerning a lack of robustness without safety consequences. Less than 10% have led to a correction in the source code. Very few issues are related to actual bugs with operational impact, which have all been also detected during a simultaneous test campaign.

The results demonstrated the validity of the approach and the ability to detect threats very hard to debug during the software development phase. Considering the reported warnings and the verification capacity, the return on investment would have made it worth applying this strategy earlier in the process. An earlier detection of safety issues would have saved costs in validation efforts.

Other considerations

The main advantage of the bottom-up strategy lies in its scalability. But it is also a way to conduct dedicated verifications on each function that is not accessible through the analysis of the whole program in one shot.

The need for this verification depends on the industrial development process. The user can be interested only in detecting runtime-errors that can really occur in its operational situation, with the whole code integrated. On the other hand, the user can be interested in verifying some properties that shall be assumed by each source function independently to address maintainability and portability concerns (that was the case of the reported UC in §5).

For example, the situation presented in Figure 5 may not impact the program if the callers of these functions are not using the non-initialised field. This situation may not be detected during the analysis of the entire program. Considering this function specifically, returning an initialized value of each output operand in any situation can be something expected and required by the coding rules of the project. In this case, the bottom-up strategy is a way of verifying these properties for each function independently.

Roles and development methodologies

During the evaluation studies, we targeted an industrial process to gain maximum benefit of the usage of static analysis in an industrial process. We targeted the phase of the development process and not a terminal phase of an independent verification.

To the question "*When?"*, the answer is as early as possible in the development process. The analysis is best run during the coding phase and before the tests. The expected benefits are a quality assessment of the source code, a reduction of the costs of the tests and a better quality of the final product.

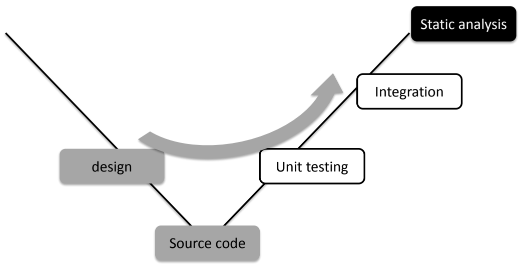


Figure 7: Formal verification of the whole product

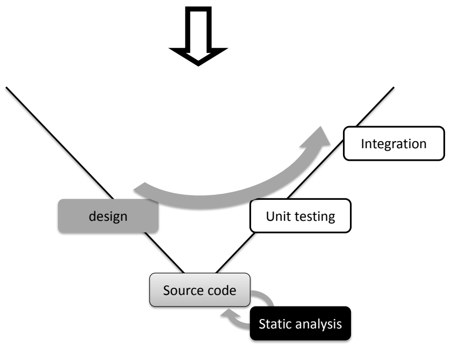


Figure 8 : Formal verification integrated in the coding phase

Another question is "*Who is doing the analysis?*". Starting from a situation where analyses are conducted by experts (Figure 7: Formal verification of the whole product), we tried to integrate as far as possible the developer in the process of verification of code by static analysis. The idea was to obtain a continuous improvement of the source code and an availability of analysis produced in a short time. Real feedbacks indicated us that the lack of skills and experience in static analysis could be detrimental. Indeed, a beginner in static analysis could be stuck on a difficulty of understanding or a weakness of the tool and spent too much time trying to get unstuck ; at worst the developer can denigrate the solution. In reaction, we quickly focused to a solution mixing skills of developers and experts in static analysis. The best way to integrate experts in static analysis in the development team is to have them prepare the verification projects and to run the first analysis. Once the project is in place, it can be appropriated by the developers. Facing problems in the analysis, experts are able to quickly find a fine tuning of the tool, a fix or a workaround and thus keep in the productivity targets. Work of experts is synchronized with the team developments by using a version control tool. In this way, we are able to keep the benefits of an integrated process and the efficiency of specialized actors.

Simultaneously to these methodological works, tooling works have been done to facilitate this application: automatic generation of stubs, module dependence analysis for retro design, makefile, and also integration in CDT Eclipse, Client/Server prototyping.

6. Conclusion

Static analysis is still a disruptive technique for verification, as it is not yet largely applied. We have demonstrated that methodological efforts can open new areas of applications. This article reports on how we applied the technique with a specific strategy and with actors that facilitated their usage in development projects. Another step would be to systematically apply these verifications during the coding stage and to capitalise on them during post-development activities, in a fashion similar to testing. If we are considering the industrial organisation, including the industrial customers, subcontractors, near-shore and off-shore, and academic laboratories, sharing the analysis projects from the first developers to the end-users can be a solution to improve quality with an optimized cost. Finally, the use of static analysis for runtime-error detection can be a first step before employing other techniques as deductive proof for functional verification.

7. References

[1] Antoine Miné and David Delmas. Towards an industrial use of sound static analysis for the verication of concurrent embedded avionics software. In EMSOFT: To appear in proc. of the 15th International Conference on Embedded Software, 2015. IEEE CS Press.

[2] Pascal Cuoq, David Delmas, Stéphane Duprat, and Victoria Moya Lamiel. Fan-C, a Frama-C plug-in for data flow verification. In ERTSS 2012: Proceedings of Embedded Real Time Software and Systems. SIA, 2012.

[3] Jean Souyris, David Delmas and Stéphane Duprat. Airbus : vérification formelle en avionique. In Jean-Louis Boulanger, editor, Utilisations industrielles des techniques formelles : interprétation abstraite. Hermes-Lavoisier, June 2011.

[4] David Delmas, Stéphane Duprat, Victoria Moya Lamiel, and Julien Signoles. Taster, a Frama-C plug-in to enforce coding standards. In ERTSS 2010: Proceedings of Embedded Real Time Software and Systems. SIA, 2010.

[5] Patrick Baudin, Pascal Cuoq, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto: "*ACSL:*

ANSI/ISO C Specification Language (V1.9)”, http://frama-c.com/download/acsl_1.9.pdf, 2013.

[6] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, Boris Yakobowski: *“Frama-C: A software analysis perspective”*. Formal Asp. Comput. 27(3): 573-609 (2015)

[7] Pascal Cuoq and Boris Yakobowski with Virgile Prevosto: Value Analysis, February 2015. http://frama-c.com/download/value-analysis-Sodium-20150201.pdf

[8] Pascal Cuoq and Boris Yakobowski with Virgile Prevosto: Patrick Baudin, François Bobot, Loïc Correnson, Zaynah Dargaye, February 2015. http://frama-c.com/download/wp-manual-Sodium-20150201.pdf

9. Glossary

ACSL: ANSI/ISO C Specification Langage

Session 24

Model Driven Engineering In practice 2

Thursday 28th, 16:30 – Ariane 2

Property Model Methodology: A First Assessment in the Avionics Domain

Patrice Micouin[1], Louis Fabre[2], Pascal Pandolfi [2]

Abstract

The aim of this paper is twofold. Firstly, it is intended to provide an overview of the goals, the concepts and the process of a new Model Based Systems Engineering methodology, called Property Model Methodology (PMM). The second aim is to provide a feedback on its application in the avionics domain. In this experiment, PMM has been used in order to develop a top level specification model regarding a textual specification of an avionics function, to validate the top level specification model, and according to PMM rules to develop (1) a design model of the function taking into account architectural constraints of an integrated avionics, (2) building block specification models and (3) building block design models. Building block specification models were validated regarding their encompassing system specification model and the selected system design model while the design models were integrated and verified, level by level up to the top level design model, regarding their specification model. This paper summarizes the lessons learnt during this process and some additional results related to safety issues. This paper, with others [1,2], proves the fundamental concepts of PMM and provides a starting point for further research on Model Based Systems Engineering of a wide range of engineered systems (discrete, hybrid, continuous and multi-physics systems), but also support additional systems engineering activities (e.g. safety-reliability activities).

Keywords

Property-Based Requirement, Property-Model Methodology, Model-Based Systems Engineering, CONOPS, VHDL(-AMS), Modelica, Simulink, Validation, Verification.

1 Introduction

There is a general agreement on the idea that there is a crisis of the classical systems engineering [3], as well as there was a software engineering crisis starting from the nineties's. Whether we consider energy, automotive and transportation or space engineering industries, the symptoms of this crisis remain the same: delivery delays, cost overruns and a lack of maturity during system's infancy. According to Nam P. Suh's vision of complexity [4], system development processes are generally too complex i.e the probability, they achieve their goals on time and for an objective cost, is too low. Among the numerous causes of this crisis we can list a cumbersome document centric approach, the technical challenges assigned to the systems under development, as well as the large, multicultural geographically dispersed teams through whom systems are developed. Although there is a shared understanding of the classical systems engineering crisis, the proposals for resolving it diverge. First, there are the pragmatists who will put forward minimum corrective actions to obtain the presumed greatest improvements. For example, the creation of best practice guides and templates is such a solution advocated by the pragmatists. Others will look into more agile methods in order to reduce misunderstandings that abound in development teams. We could designate them as inter-subjectivists ("*people rather than processes*[1]"). Finally, there are those

[1] http:/www.agilemanifesto.org/sign/display.cgi?ms=000000309.

[1] Patrice Micouin
Arts et Métiers ParisTech, LSIS, UMR CNRS 7296,
2, cours des Arts et Métiers, 13617 Aix-en-Provence, France
patrice.micouin@incose.org
[2] Louis Fabre, Pascal Pandolfi
Airbus Helicopters
Aéroport Marseille Provence
13 700 Marignane France
{louis.fabre, pascal.pandolfi}@ airbus.com

who see a solution in rigorous formal processes. Although we consider that each approach contains a grain of truth and deserves to be explored, we undeniably side with the last one. Indeed, we claim that a formal development process including validated specification models and verified design models could solve the classical systems engineering crisis, whose principle was designed decades ago [5], by limiting the number of possible misinterpretations [1]. In this paper, (1) we shall present the Property Model Methodology (PMM), then (2) we will offer some insights into its utilisation through the case of an avionics function. Lastly, (3) we will summarize the main lessons learnt in this experiment and we will identify the way to go before PMM can be deployed operationally.

2 PMM: goals, processes and concepts.

PMM is an innovative model-based systems engineering (MBSE) method [6] focusing on operational goals. It is a top-down approach that authorises the re-use of pre-existing blocks at any hierarchical level of a system model's architecture. Another feature of PMM is that it complies with current industrial development standards, specifically ARP4754A [7] and EIA632 [8]. PMM is a method, derived from the scientific method [9, p 10-11] and applied for engineering systems, i.e. a sequence of operative rules, defined to build specification models and design models of engineered systems and expressed in a language. Finally, the third pillar of PMM is simulation, which is the primary means for validating specification models and verifying design models. The languages that can support directly PMM are simulation languages such as VHDL [10], VHDL-AMS [11], Modelica [12], Simulink [13] and the PMM concepts are directly mapped onto simulation language features to produce simulation models. Today, PMM has no connection with languages such as SysML [14] or Domain Specific Languages, while the OMG[2] is currently analysing the need to model requirements more clearly in SysML[3,4].

Roughly described, for an innovative system, PMM usually starts just after the validation of a Concept of Operations [15] and is made up of 4 activities[5]:

(1) The first activity, which is carried out at the system level, is the development of a goal oriented system specification model associated with simulation scenarios defined in order to validate the system specification model. First innovation of PMM: it supports an early validation process based on simulation, before starting a solution definition process.

(2) Then, we continue with a recursive process that consists in breaking down the system into building blocks. Each building block is made up of a specification model and one or more design model alternatives. Thus, a collection of competing system's architectures is considered, but only the preferred one is selected. The building blocks and their connections constitute the system's architecture. At each hierarchical level, the building block specification models are validated together against higher level building block specification model from the system specification model to the lowest level. A building-block is qualified as elementary when it is not decomposed any further and when its behaviour, is completely defined by a Behavioural Design Model (BDM) [6, p 133]. Libraries store the building blocks that might be re-used in future developments. From the point of view of its acquirer, any building block that is picked from a pre-existing library is also considered as an elementary building block. The design process ends when all the elementary building blocks have been identified, and modelled or acquired. The PMM recursive design process

[2] OMG = Object Management Group : http://www.omg.org/

[3] http://www.omg.org/issues/issue17016.txt

[4] http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-roadmap:requirements_modeling_working_group

[5] It also includes recovery points to reengineer the system when a goal is not met.

is, by several aspects, a very classical process. It is acquainted with different design methods such as the Suh's Axiomatic Design method [16], the Abrial's B method [17].

(3) The third activity, after this top-down design process, is a bottom-up process that consists in recursively integrating the elementary building blocks and then the intermediate building blocks so as to finally end up with the complete system model. For each integration level, design models are integrated and verified against their own specification models. Second innovation of PMM: it supports a design verification process based on simulation before starting the physical implementation.

(4) The final steps that deal with the production of the physical building blocks (Hardware and Software), their integration, and the final installation of the system in its environment are out of scope of this paper, but we encourage readers to read [6, chap. 11] for further details.

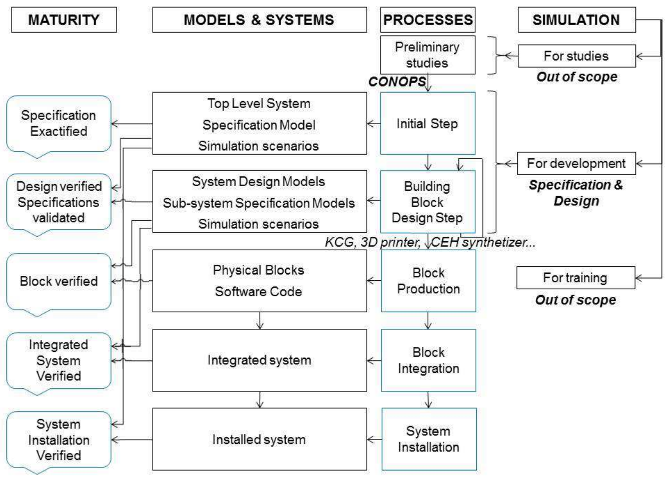

MATURITY
MODELS & SYSTEMS
PROCESSES
SIMULATION
Preliminary studies
For studies
Out of scope
CONOPS
Specification Exactified
Top Level System Specification Model Simulation scenarios
Initial Step
For development
Specification & Design
Design verified Specifications validated
System Design Models Sub-system Specification Models Simulation scenarios
Building Block Design Step
KCG, 3D printer, CEH synthetizer...
Block verified
Physical Blocks Software Code
Block Production
For training
Out of scope
Integrated System Verified
Integrated system
Block Integration
System Installation Verified
Installed system
System Installation

Figure 1 PMM processes.

2.1 PMM Specification Process and Specification Models.

Engineered systems are goal oriented. If they are correctly developed, they embody the intent of their designers. This is the reason why system specification is the first activity in system development. PMM proposes a Goal Oriented Requirement Engineering approach such as KAOS [18]. The specification modelling sequence is the third innovation of PMM. This is a very unusual backward process from the effects to the causes: (1) first of all, the specification process starts with the identification and definition of system goals. Goals are modelled as outputs of the specification model. (2) Then the actualisation conditions of these goals are identified. These conditions are modelled either as observable states (a special kind of PMM outputs) or as expected inputs. (3) Finally, the result of the analysis is formalized as one or several PBRs [19]. Comes the fourth innovation of PMM: PBRs are predicates that link goals, observable states and inputs in order to specify the actualization conditions of system's properties. The PBRs structure prevents introducing design biases at specification level. The same process applies for secondary[6] (undesired) outputs such as system failures and inputs (expected or undesired). The PBR theory is based on the theory of properties [20] due to the Canadian epistemologist Mario A. Bunge.

[6] According to a terminology due to Vladimir Hubka and Ernst Eder in [21]

The basic form of a PBR is as follows:

PBR #n : when C → val(O.P) ∈ D

This formal statement means: "*when the condition C is true, the property P of the object O is actual and its value shall belong to the domain D*" where C is a relevant condition for the system or its environment: a functioning mode, a system state, an (undesirable) event, a time delay or a combination of such features and where the domain D is a finite or infinite set such as {on, off} or $\mathbb{R}^n$ (possibly linked to a frame and a physical unit).

The concept of a PBR can be implemented as a conditional assertion (Boolean function) with various simulation languages such as those mentioned above.

Because they are outside the system's developer control and only presumed, assumptions are specific PBRs limited to input properties.

Based on Bunge's property algebra, several PBRs can be combined thanks to the conjunction operator "$\wedge$" in order to build composite PBRs. In a dual way, the partial order relationships "$\leq$" and "$\geq$" enable analysts to compare two PBRs. Thus, the expression "$PBR_1 \wedge PBR_2$" is the conjunction of PBR_1 and PBR_2 and is itself a PBR. Moreover, the statement "$PBR_1 \leq PBR_2$" means PBR_1 is less constraining than PBR_2.

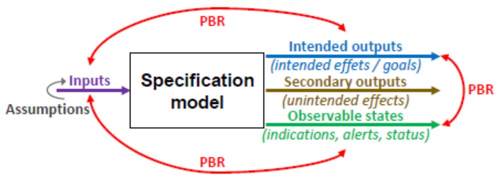


Figure 2 PMM Specification model

A specification model is a formal model that includes (1) system requirements, (2) system interface requirements and (3) system assumptions. The expression of specification models into simulation models (VHDL-AMS, Modelica or Simulink) is a solution to make sure that specification models and interfaces are coherent. In addition, the simulation of a specification model linked with the corresponding equation design model (EDM) [6, p 139] provides analysts with various advantages. First, it is an assistance to guarantee the completeness and correctness of the top level system specification model for a given set of validation scenarios. Second, it provides also a visual means to assist stakeholders such as pilots, flight engineers, authorities in the validation of the top level system specification model before starting a design on the basis of a specification model presumed free of errors. Third, as we will explain, simulation provides also capabilities for validating building block specification models against higher level building block specification models up to the system specification model. This is also an innovation of the PMM process which leads us to the "Prime contractor theorem" described below.

2.2 PMM Design Process, Design Models and PBR Derivation

Once the system specification model is complete and correct, the second system development activity consists in developing a system design model. This is a design process very similar with the Suh's Axiomatic Design "zigzagging process" between specification and design models. For a very simple system, the system design model is usually an EDM. However, for more complicated systems, Structural Design Models (SDMs) [6, p 145] are introduced. A SDM is a formal definition of a system's architecture $\mathcal{A}$ that is made up of three elements: (1) building blocks; (2) an endo-structure linking together the building blocks that belong to the system; and (3) an exo-structure linking together system building blocks with objects that belong to the environment, through its interfaces.

For each candidate structural design model, the third system development activity consists in deriving the system requirements {PBRs} into building-block requirements {PBR_1, …, PBR_n}. To be valid, for a given system structural design model $\mathcal{A}$ and for a set of assumptions on properties own by objects that belong to the environment $\mathcal{EA}$, the conjunction of the derived building-blocks PBR_1, …, PBR_n shall be more constraining than the system PBRs. This statement can be expressed formally as:

$$\text{Derivation (PBR): when } \mathcal{A} \wedge \mathcal{EA} \rightarrow PBR \leq PBR_1 \wedge \ldots \wedge PBR_n$$

The PMM uses simulation as the main method to validate the derivation of system PBRs into a set of derived building block PBRs with respect to a set of validation scenarios (Fig. 3); The extend of the set depends on the level of validation rigor.

This validity condition of PBR derivation leads to the "**Prime contractor theorem**":

"*A sufficient condition for a system to comply with its PBRs is that its building blocks comply with the PBRs validly derived from the system PBRs, provided the design choices and assumptions made about the environment driving the derivation remain valid.*"

This result of PMM, shared[7] with the B method, is a key point for a prime contractor to secure his subcontracting process.

The design process – design, derivation, derivation validation – is applicable at any level within the system's architecture, down to the lowest hierarchical level corresponding to the elementary building blocks. Lastly, the design verification is performed according to § 2 (3).

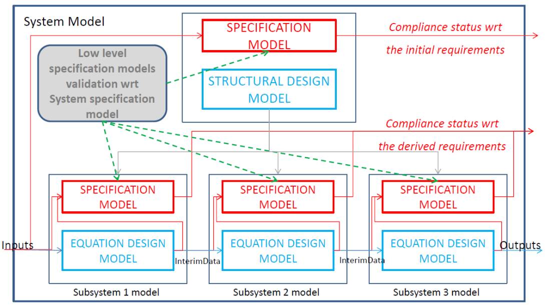


Figure 3 **Validation process of subsystem specification models**

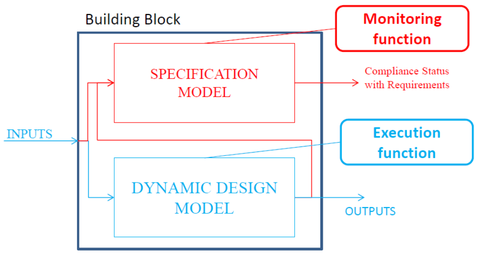


Figure 4 **Specification and design model roles during a verification process**

Simulation also provides capabilities for verifying design models. While the simulation is running, for all submitted simulation scenarios, Fig. 4 shows that the specification models monitor the interacting design models so as to check whether any requirement is violated. If there is no violation detected during the verification of a building block design model, then the building block design model may be presumed as free of design errors.

To complete the picture, the PBRs theory provides us with a theorem of practical importance, the "**Safety theorem**":

"*The introduction of any additional PBR at any building-block level of a system does not impair the safety of this system. It can impair its feasibility.*"

[7] However, keep in mind the differences between evidences provided by simulation and formal proofs.

Obviously, this theorem, established in the frame of the PBR theory, is wrong in the context of textual requirements.

3 "Clearance Test Function" experiment

The case study comes from the avionics domain and is based on an avionics function called "Clearance Test Function" (CTF). This is a limited experiment selected in order to prove the PMM concepts and to identify benefits, limitations and possibly improvements. The VHDL language was used as target simulation language. Specification and design models were written in VHDL and the ModelSim® simulator from Mentor Graphics was used. Note that a translation schema of PMM models in VHDL models is provided in [22].

3.1 Experiment context

The main goal of the "Clearance Test Function" is to provide the crew with a status of various systems and equipment installed inside the helicopter, before taking off. In conjunction with the MMEL[8], it provides to the crew with an aid to clear or not the flight depending on the availability and the healthy status of systems and equipment such as sensors, actuators, control units, etc.. The "Clearance Test Function" is activated on ground when the helicopter is powered on. It includes several sequences of tests, triggered automatically or upon crew request. The function consists in the reporting of synthetized messages for the crew computed on the basis of PBITs, CBITs and IBIT[9]s produced by the systems and equipment involved and H/C or avionics states. The "Clearance Test Function" is a decision making aid for the crew.

At the time of the experiment, this function was under development in the frame of a new avionics suite developed by Airbus Helicopters and called "Helionix®". It is now installed on several in service helicopters such as the H175, H145.

At the beginning of the experiment, (1) a specification including 200 textual requirements and (2) a preliminary Rotorcraft Flight Manual (RFM) describing pre-flight tests were available and were foreseen to be used as main inputs for the experiment. Additionally, functional and physical architectural choices of Helionix® avionics suite, were well known at this time.

3.2 PMM application

This paper describes the PMM application, but practical examples cannot be disclosed in this paper. Disclosable pieces of examples will be part of the conference presentation.

Specification:

After an initial learning phase of the input data, the first step of the experiment consisted in the establishment of the top level PMM specification model of CTF. With the objective to start from the operational need, the RFM was demonstrated to be a better source than textual requirements to produce the PMM specification model because it was really representative of operational need (as a CONOPS would represent this need).

In order to build the top-level specification model, the messages indicated in RFM were inventoried and grouped depending on the progress in the test sequence. These groups were modeled as intended outputs of the specification model. The possible values for each output corresponded to a specific message displayed to the crew.

[8] MMEL=Master Minimum Equipment List

[9] (P=power-up, C=continuous, I=initiated)BIT=Built-in Test

After this phase of identification of expected outputs, their actualization conditions were analyzed. For instance, the analysis provided us with the following conditions: the message “Pump xx failed” (intended output) shall be displayed to the crew 2 sec at most after a “Pump test” (modeled as observable state) request sent by the crew when the following conditions are fulfilled: the helicopter is on ground (modeled as observable state), one engine at least is started (modeled as observable state), and a Fail IBIT status from Pump xx” (modeled as input) is received by the avionics.

The same analysis was performed for each specific message and for each group. Although somewhat tedious, it was easy to represent these conditions as PBRs or, what amounts to the same, as boolean functions. Translation errors were possible but would be detected during a sufficiently rigorous validation (obviously this event occurred).

After each intended output associated with its PBRs, the same procedure was applied to the observable states in order to specify the actualization conditions of observable states.

Finally, assumptions about the inputs (values, frequency, sequencing, ..) were also modeled as PBRs, completing a first version of the specification model of the CT function.

This PMM specification model was made up of 14 property based requirements (PBR) prescribing completely the external reactions of the CTF function to handle about 200 external events coming from its environment and also few dozen internal events. It was not necessarily consistent with the real need, it could contain specification errors but it was formally coherent and formally complete, which is rarely the case for textual specifications.

Validation:

It is the responsibility of the validation process of showing that the specification is as exact as possible, that is to say, it characterizes the right system i.e, the expected system. Although tedious, in PMM, it is quite easy to build validation scenarios from the expression of PBRs and this should be largely automated. This ability to automatically generate validation cases is another feature of PMM validation process. We have built a simple scenario to validate the specification model.

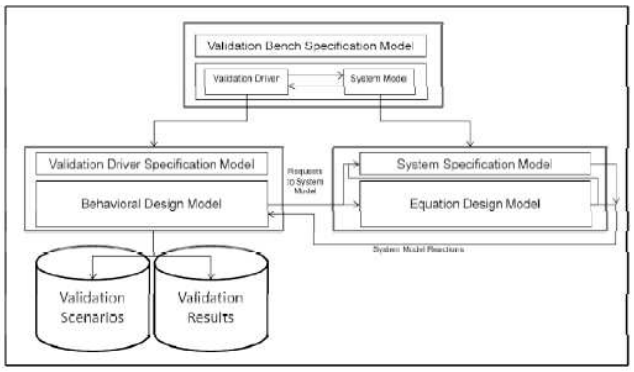


Figure. 5. PMM validation process of a specification model.

The next step was to transform the CTF Specification Model into an Equation Design Model according to the PMM process. As expected, the experiment showed that this transformation was direct, but it was handmade for lack of available transformation tool. The CTF system model, comprising its top level Specification Model and its equation design model was embedded in a validation bench with a validation driver in order to perform the validation process of the CTF top level Specification Model by the means of simulation (Fig. 5.). This validation process, performed with a moderate effort, showed that the CTF system model outcomes were in line (after some corrections) with the preliminary RFM.

Quickly, the PMM specification model of CTF became a precise representation of the observable behavior of CTF, reflecting expected effects and their conditions of actualization as they were described in the RFM, while staying absolutely free from any design choices to implement the function. On the basis of this validation process, the CTF Specification Model was considered as validated (as exact as possible), i.e. in line with the right function.

Fourteen property based requirements were defined to describe the complete expected behavior of CTF. However, more than the significant reduction of requirements, which is not, in itself, a good indicator, what should be noted is (1) the accuracy of PBRs, (2) their formal coherence and formal completeness, (3) their ability to be derived in cases and scenarios of validation and verification and (4) their lack of design bias. They allow very firmly separating what concerns the specification of which is within the design.

Design and PBR derivation:

The next step was to model a PMM Structural Design Model (SDM) of CTF consistent with the Helionix® avionics system architecture in terms of behavioral and physical breakdown. The CTF SDM had to fit the Helionix® integrated architecture, its hardware and software components and the space and time partitioning of its resources. This implementation was time consuming, it was not held back by any theoretical or practical issues, but by the hand-made and boring writing of the CTF models. This activity showed that it was possible to submerge CTF function design model in the avionics system design model, and to comply with the constraints of its architecture, respecting its breakdown and the allocation of resources according to the defined temporal scheduling.

In the case, the derivation of PBR was not an issue; every component has been assigned a contribution to satisfy the mother PBR. Obviously these derived PBRs (in accordance with ARP4754A's derived requirement definition) were not directly traceable from the mother PBR and depended on Helionix®'s architecture choices and would be very different in the context of a federated architecture, for example.

As a first consequence of the use of a simulation language such as VHDL, the various configurations of models built during the development process were formally coherent and formally complete. In particular, the interfaces of all the building blocks composing each CTF system model configuration were coherent and complete.

Verification:

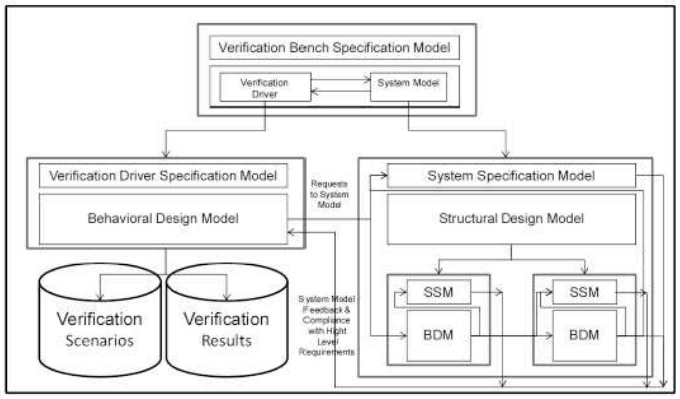

Verification Bench Specification Model
Verification Driver
System Model
Verification Driver Specification Model
Behavioral Design Model
Requests to System Model
System Specification Model
Structural Design Model
SSM
BDM
SSM
BDM
Verification Scenarios
Verification Results
System Model Feedback & Compliance with Hight Level Requirements

Figure. 6. PMM verification process of a design model.

As fourth step, the CTF system model, comprising its top level Specification Model and its SDM was embedded in a verification bench with a verification driver in order to perform the verification process of the CTF System Model by the means of simulation (Fig. 6.).This verification process, performed with a moderate effort, showed that the CTF design model was in line with its specification models (after several corrections). For the performed effort of verification, the SDM was deemed error free. Deadline requirements were satisfied by the temporal scheduling (with assumptions on worst case execution time). During this experiment, we found that it was possible to automatically generate simulation cases from the structure of PBRs depending on the level of criticality of the system considered. These simulation cases put together in simulation scenarios were used as validation scenarios for specification models and verification scenarios for design models. These simulations scenarios could also be re-used for the testing of physical products [6, p 162-167]. Thus, PMM allows, on the specification models basis, to size early testing effort which must be performed and almost automatically generate scenarios corresponding to the required effort.

Safety considerations:

The final step of the experiment consisted in specific investigations about safety requirement modeling and the verification of the efficiency of design mechanisms to prevent failures. The experiment allowed to verify whether a given architecture could tolerate faults or not, thanks to a special device of PMM, the RDM [6, p 207]. The Reliability Design Model (RDM) is a design model derived from a structural design model (SDM) that allows us to observe the behavior of a system model in the presence of single or combined faults. It has been shown that, for a moderate level of verification rigor, a system model considered in the experiment properly met the requirement "*no single failure will result in a catastrophic condition*" according to AC29-2C, F-17 [23].

4. Lessons learnt and conclusions

The experiment described above was conceived as a proof of concept and it is not excessive to say that the Property Model Methodology concept has been validated beyond the initial expectations. Not only, all the expected confirmations concerning the relevance of PMM concepts and PMM methodological process have been obtained but further results were achieved.

First of all, the experimentation established the suitability of the concept of PBR and showed that the process of specification associated therewith allows the production of Specification Models without design bias. Then, the experiment confirmed that the Equation Design Models allowed the validation of specification models and showed that the former could be produced automatically from the latter. It confirmed also that the validation and verification cases could be automatically generated from the structure of PBRs with a gradation depending on the level of rigor requested.

The experiment showed that it was possible and quite easy to design a CTF solution that was consistent with architectural choices made for reasons (Helionix® architectural principles) beyond the specific needs of the developed function.

The last but not the least, the experimentation has highlighted several possible non-recurring cost (NRC) savings in the development of a system: (1) a system model formally coherent from the top to the end building blocks and free of interface errors (2) specifications interpretable as little as possible while misinterpretations are source of errors and waste of time, (3) early requirement validation through simulation before any physical (hardware or software) realization of the products composing the system, (4) a design verification through simulation and, in particular, of failure prevention mechanisms, (5) automatic generation of unit test cases, integration test cases for the physical system and its building blocks and installation test cases of the system into its environment.

However, despite these results, the deployment of PMM is not immediately feasible. Indeed, during the experiment, the models were produced by hand by an engineer familiar with VHDL. This pre-requisite is not acceptable if PMM is applied by engineers from different engineering disciplines. More, existing simulation tools are design oriented, bottom-up oriented and ignore specification aspects and, a fortiori, the theory of PBRs. So, a PMM front-end, hiding the underlying language for modeling PMM artifacts in a way acceptable by discipline engineers, and especially for modeling PBRs and for producing specification models, remains to be developed. This objective is at hand regarding simulation languages such as Modelica [2] and Simulink. It remains to be determined regarding VHDL and VHDL-AMS. An alternative way to this approach targeting a particular simulation language is of a PMM Modeler based on a pivot language and capable of generating models expressed in various simulation languages. This is the way that the Ellidiss company choses to explore thanks to its LMP technology [24].

References:

1. Poupart E, Wallut J.M, Micouin P, Property Model Methodology: A First Application to an Operational Project in the Space Domain, CSD&M 2015, November 2015.
2. Pinquié R, Micouin P, Véron Ph, Segonds F: Property Model Methodology: a case study with Modelica, to be published.
3. Newport, J.R.: Avionic Systems Design, CRC Press (1994)
4. Suh N.P, Complexity, Theory and Applications, Oxford University Press, 2005.
5. Hall, A.D., (1962), A methodology for Systems Engineering, Van Nostrand, Princeton, New York.
6. Micouin, P.: Model Based Systems Engineering: Fundamentals and Methods, Wiley & ISTE (2014).
7. ARP4754A, Guidelines for Development of Civil Aircraft and Systems, SAE, Warrendale (PA), 2010
8. ANSI/EIA632, Processes for engineering a system, GEIA, Arlington, VA, 2003.
9. Bunge M., (2007), Philosophy of science, volume 1: from problem to theory, Chapter 1, The scientific approach, Transaction Publishers, New Brunswick, New Jersey, 4th print.
10. IEEE Standard VHDL, (2008), Language Reference Manual, IEEE 1076, IEEE Computer Society.
11. IEEE Standard VHDL Analog and Mixed-Signal Extensions, IEEE 1076-1, IEEE Computer Society, 2007.
12. Modelica Association, Modelica® - A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.3 May 9, 2012.
13. Karris, S.T: Introduction to Simulink with Engineering Applications, Orchard Publications, 2011
14. OMG Systems Modeling Language (OMG SysML™), Version 1.4, September 2015, http://www.omg.org/spec/SysML/1.4/
15. IEEE Standard 1362, IEEE Guide for Information Technology – System Definition – Concept of Operations Document, 19 March 1998.
16. Suh N.P, Axiomatic Design, Oxford University Press, 2001.
17. Abrial J-R, Modeling in event-B, System and Software Engineering, Cambridge University Press, 2010.
18. Lamsweerde von, A. Requirements Engineering**:** From System Goals to UML Models to Software Specification, Wiley (2009).
19. Micouin, P., (2008), “Toward a property-based requirement theory: system requirements structured as a semilattice”, in: Systems Engineering, Vol. 11-3, pp. 235-245.
20. Bunge. M, Treatise on Basic Philosophy, vol 3, Ontology I: The furniture of the World, D. Reidel Publishing Compagny (1977).
21. Hubka V, Eder W.E: Theory of Technical Systems: A Total Concept Theory for Engineering Design, Springer-Verlag, 1988.
22. Micouin, P., (2014), “Property-Model Methodology: A Model-Based Systems Engineering Approach Using VHDL-AMS”, in: Systems Engineering, Vol. 17-3, pp. 249–263.
23. Federal Aviation Administration, AC29-2C Certification of Transport Category Rotorcraft, latest version.
24. Dissaux P., Hall B. Merging and Processing Heterogeneous Model, to be published, ERTSS 2016

MDX and AUTOSAR Standards for Model Sharing to leverage Tier1 - OEM cooperation in the ECU software development

Stéphane Louvet[1], Dr. Mouham Tanimou[2]

[1] Robert Bosch (France) SAS, Diesel Gasoline Systems, Electronic Control, 32 avenue Michelet, 93404 Saint-Ouen, France
[2] Robert Bosch GmbH, Diesel Gasoline Systems, Electronic Control, Postfach 30 02 20, 70442 Stuttgart, Germany

Keywords : Engine Control, embedded software, Model Based Design, Software Sharing, Model Sharing, MDX, AUTOSAR, automatic code generation, ASCET, Simulink®

Abstract / Motivation

Software Sharing in automotive embedded software development has continuously grown over the last decades and is still getting more importance and attention. Main advantages for Software Sharing, as seen by the OEM, are acceleration of development time and cycle as well as a full flexibility to customize the final product, with introduction of its own software.
Although classic software sharing has shown its capabilities in practice, it shows some limitations and need to be extended in order to cover additional use cases. The concept of model sharing can help to address those use cases, the following paper outlines the use of ASAM-MDX and AUTOSAR standards with the model sharing to enhance customer cooperation.

1. Introduction

An increasing number of car manufacturers develop their own functionalities and/or their own software platform in order to communize functions across several control units supplied by different Tier 1 companies. This happens in general as stub-software - already compiled - to be integrated in the final Software. This proven process in practice needs however to be improved and can thus be enhanced to increase development efficiency.
In order to achieve such goals, OEMs need the ability to test enhanced or newly developed functions in a very early stage of development, those function checks need to be performed continuously along the V-Cycle development up to the final software. Model Based Design is a general accepted mean to do this and is becoming widespread in the automotive industry.
Within the automotive industry, modeling-tools like Simulink® (together with Embedded Coder or TargetLink® as code generators) and ASCET are widely used by OEMs to verify their functions in an early phase using virtual and/or rapid prototyping methodologies. The so verified model together with the achieved data specification can then be exchanged with suppliers as executable specification for functions to be integrated in the ECU; this is one side of the medal of what is called “Model Sharing” at Bosch. The other side of the same medal consists of provision of a development environment for simulation as well as for software build, together with a set of common modeling guidelines and libraries and common data description standard for further development.
This approach has a high degree of flexibility, since it provides a framework for OEMs starting from the scratch, but also allow to adapt the derived Process, Method and Tools to car manufacturers having their own approaches. Bosch has established with this approach a five-level classification for model Sharing between Tier1 and OEM, which starts from "Use of ECU virtualization environment” and goes up to “Joint Development”.
Use of Standards for the data description (e.g. ASAM with -MDX, CDF …- and/or AUTOSAR with service libraries, methodology …) as well as standardized interfaces specifications (e.g. as published with AUTOSAR R4.x) can significantly increase the effectiveness of this approach; e.g. by easing labels data management and exchange between development partners through a database (e.g. Visu IT!-ADD), as well as exchange of models.
The use of standard for data description with the approach described above raises following central questions that are subject to further analysis in this paper:

- How can standards be used with ECU Software development tools?
- What are the constraints on modelling pattern?
- How can migration of models be handled/managed between different standards?

In this paper we will present a rough description of benefits coming with the standards ASAM-MDX and AUTOSAR with respect to mutual development cooperation. Furthermore, we will focus also on their deployment with modeling tools; the technical aspect to ensure a seamless migration from MDX to AUTOSAR standard will be also described. Some of the results as well as benefits achieved with these standard in customer cooperation within series projects will also be presented.

2. Model-Sharing

Model-Sharing is a way of exchanging models and model artefacts in order to ease Software development and cooperation between two or several partners; it is part of our Model Based Software Development framework in the ECU development domain. It is also a way to allow rapid prototyping as well as simulation.

Basically, motivations for Model Sharing in embedded SW development process are:

- OEM focuses on physical modeling, simulation and rapid prototyping. The OEM can also, via model exchanges, take benefit from supplier physical modeling and simulation know-how.
- The supplier is in charge of industrialization of provided models as well as code generation from these models.

Provided models (In general in form of ASCET, Simulink® or TargetLink® models) by the OEM can be used as executable specifications for Software development by the supplier. Using a common tool with it a common language, and standardized files by the development partners for modeling leads to reduction of the amount of to be exchanged documents, and thus enhance the understanding of OEM requirements by the suppliers. This can significantly lead also to reduction of development effort and development time as well as delays in the development.

Typical use cases for Model Sharing are described in the sections below.

<u>Use case 1:</u> OEM describes the expected functionalities in customer specific functions developed by the supplier

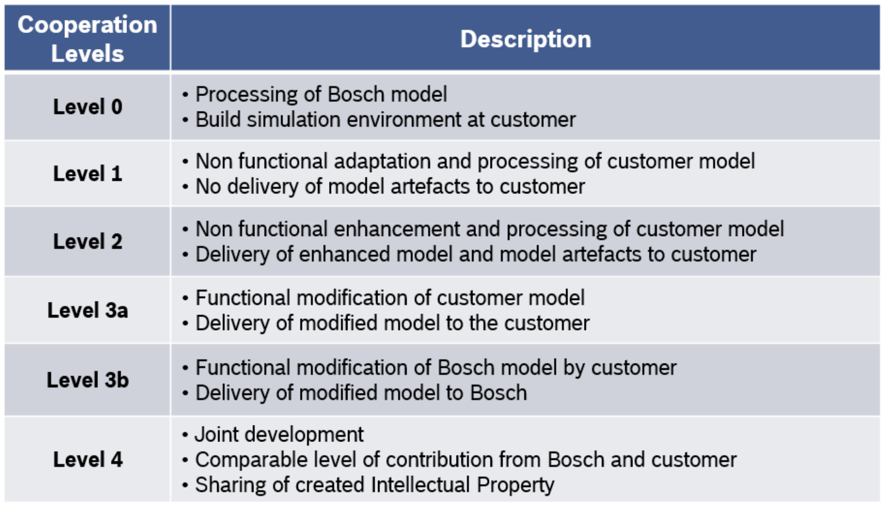
- the desired changes can be implemented directly in the model
- Simulation/Rapid Prototyping can be carried out w/o waiting for an “official” SW release

<u>Use case 2:</u> OEM develops its owns model and the supplier is in charge of the “so-called” industrialization of the code

- This corresponds to the so-called “Commissioned Development” in Bosch process development
- The code implementation is simplified if both partners are working with the same modeling environment (examples : ASCET, Simulink®, TargetLink®)
- Functionalities can be tested with Rapid/virtual Prototyping before delivery to the supplier
- corrected (enhanced) model can be delivered back to the OEM: This helps to avoid additional traceability documents and minimize risk of forgotten improvements in own model

Cooperation levels

Since each customer has its own dedicated development process, it has therefore specific requirements for development cooperation with suppliers. On the other hand, suppliers need to classify the requests of OEM to find a common approach. Following this line of thought a five-level classification for model Sharing between Tier1 and OEM has been performed at Bosch: from "Use of ECU virtualization environment” up to “Joint Development” [3]. This build a base for discussions with OEMs before starting a new project and it provides a framework for development methodology and contracting. The cooperation usually contains aspects of different types of contracts like service or licensing contract. However, these aspects are typically covered by one agreement describing the complete scope of the cooperation.



Cooperation Levels	Description
Level 0	• Processing of Bosch model • Build simulation environment at customer
Level 1	• Non functional adaptation and processing of customer model • No delivery of model artefacts to customer
Level 2	• Non functional enhancement and processing of customer model • Delivery of enhanced model and model artefacts to customer
Level 3a	• Functional modification of customer model • Delivery of modified model to the customer
Level 3b	• Functional modification of Bosch model by customer • Delivery of modified model to Bosch
Level 4	• Joint development • Comparable level of contribution from Bosch and customer • Sharing of created Intellectual Property

Figure 1 - Different cooperation levels for model sharing

The picture below illustrates those cooperation levels in a contracting and process flow view

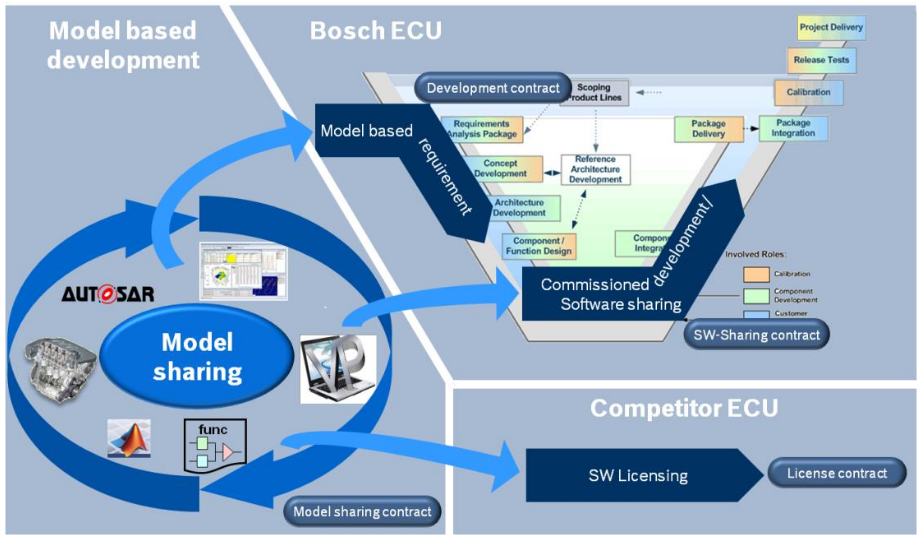


Figure 2 - Workflow for Model Sharing and contracting

Some of the cooperation models will be shortly described hereafter to better understand the use cases which are addressed in this paper.

Level 0: Virtual Prototyping

Since several years, it can be observed that the trend for ECU software engineering is moving towards PC simulation in order to

1) Save costs by increased development productivity and
2) Manage complexity and apply holistic engineering approaches.
3) Allow different development speed between HW, mechanics and SW

And OEM are requesting from Tier1 supplier a stringent support of development cycle oriented use-cases at OEM in a virtual manner. This requires to build a virtual test bench, which contains virtual engine ECU-Software (as it is running in the real ECU) as well as representative plant models in a virtual environment, which are real time capable. For the virtual test bench, PC-executable ECU-functions in a powerful simulation environment are then needed [1] [2].

Level 1: Processing of OEM Model

This level of model based cooperation is the so called “Commissionedd Development” within the development process at Bosch; the OEM delivers a (simulatable) model that has to be processed by the Tier1 supplier to generate production code without any change on the model. Alternatively, the OEM

can also deliver a specification (e.g. PDF-document) and the supplier creates a model according to this specification. This model will then be delivered to the OEM for further development.
Using same modeling tool by the OEM and the Tier1 supplier helps to avoid converting the model to another modeling environment.

Level 2: OEM Model Enhancement and Processing
This model sharing cooperation level is also considered as part of the development process "Commissioned Development" at Bosch.
The OEM delivers a model and the Tier1 supplier can introduce non-functional necessary modifications in the model for several purposes, such as:

- optimisation for production code generation
- model corrections in case of mistakes found during model implementation
- specific interfaces e.g. for diagnostic

In comparison to cooperation Level 1, the final model contains added value introduced by the Tier1 supplier to the original the OEM model. Furthermore the development efficiency is getting enhanced, since the modified model by the supplier could be redelivered to the OEM and thus modifications done by the supplier are then available to the OEM and so risks of model discrepancies are reduced since the corrections are done at a single site.

Level 3:
Level 3a: Use of OEM Models in Cooperation
The OEM owns a model and delivers it to the supplier who is then charged to industrialize the model and generate production code generation out of the model. Different to cooperation Level 2, the supplier can introduce functional modifications in the OEM model. The kind of Know-How added by the supplier can be for example specific calculations mechanisms such as special filters (e.g. replacement of PT1 filter by a Butterworth filter to achieve certain goals). The modified models are delivered back to the OEM for further development. In general, for all cooperation levels, a CIL (Customer Interface Layer) is needed to combine OEM functions with Bosch functions.

Level 3b: Use of Bosch Models in Cooperation
The supplier provides its own models to the OEM who can then enhance the model by its own functionalities and Know-How. The OEM delivers back the modified model to the supplier who is then in charge of industrialisation and production code generation. The final responsibility for the model (physical Behaviour as well as code out of the model) remains with the supplier who should review the OEM functional modification for approval. The development time of OEM specific functions can be drastically reduced with this cooperation level since there is no need for the OEM to deliver any specification or schematic that would have to be analysed by the supplier and then interpreted in the model.

Although Know-How/functionality is appended to the original model with this cooperation level, the ownership of the model remain with the original provider of the model; i.e. Ownership of model remain with OEM in case of cooperation level 3A and ownership of model remain with Bosch with cooperation level 3B.

Level 4: Joint development
The joint development deals with a common ownership of the models. The base model can be created at the start of the partnership or can be provided by one of the party. The level of contribution from the supplier and the OEM is comparable and the created Intellectual Properties are shared. It leads to accelerated development: a unique model can be continuously exchanged between the supplier and the OEM or modified by one party with the contribution of the other party. As a constraint, it is necessary to define clear responsibilities between the supplier and the OEM.

3. Standards with Model-Sharing cooperation

ASAM-MDX standard and AUTOSAR standards are being considered in this paper.
The ASAM-MDX standard is an ASAM (Association for Standardisation of Automation and Measuring systems: http://www.asam.net) standard, and the signification of MDX is "Meta Data Exchange Format". This format is often used with another ASAM standard called CDF (Calibration Data Format).
AUTOSAR stands for "AUTomotive Open System ARchitecture": http://www.autosar.org. This standard has been released by a consortium of car makers, software suppliers and tools suppliers. This standard describes also data specification for development collaboration.

Both standards (ASAM-MDX as well as AUTOSAR) describe the interfaces and data definition used in the corresponding models within XML files. The AUTOSAR standard is more complete since it describes also the Basic Software as well as the interfaces between the Application Software and the Basic Software. Both standards can not be applied at the same time on a model: the code can be either generated with a MDX target or with an AUTOSAR target.
The most evident advantage of such standard remains in the fact that the data descriptions can be easily shared between OEM and suppliers at the same time as the models exchange. Since the format used is XML-language, these files can be processed more easily than Excel-like formats and are less sensitive to characters issues. Another advantage of these standards is that it helps to avoid developing specific tools or scripts as it is the case when using specific OEM or supplier formats. Development activities can then be performed with standard tools from the market without any deep customization.

As generally known, a function design can be performed by using different modelling styles. Hence standards may have an impact on the modelling pattern. The cooperation between OEM and suppliers with models exchanges can be eased if both partners are using the same standard for their Model Based Development tool chain. Non-use of same standard can really impede model exchange, especially with regard to library blocks intended for simulation as well as for code generation; some library blocks may include memory states like for filters or flip-flop, and thus the memory states can be described with workspace elements in form of Simulink® objects or Matlab objects.

With R2012B onwards, MathWorks has introduced with a PSP (= Pilot Support Package) in Simulink® a set of MDX objects (derived from Simulink® objects) which have been developed in cooperation with Bosch, intended for use in cooperation projects between Bosch and its customers. Together with this PSP, Bosch has developed a Simulink® library containing AUTOSAR services and other specific functions (e.g. BSW specific functions). In case of Simulink® model exchange containing these library blocks, a simulation can be performed since these objects are based on Simulink® objects; code generation will however require the use of the MDX package. If however the data specification on customer site are based on Matlab objects, simulation will be performed with default values but not with the specified version; a conversion will therefore be needed to have appropriate simulation. Furthermore, working with a standard for the data description avoids also to create its own specific data description rules. It then avoids to create its own scripts to generated the data description files during the code generation phase.

From a modelling point of view, a standard like AUTOSAR provides also standardized interfaces with the Basic Software like diagnostic interfaces, and thus permits to connect easier Application Software functions to the Basic Software services.

From an integration point of view, if the OEM provides Software Sharing code which has been generated with standardized interface, the integration of the Software Components in the supplier Software is made easier since the files can be processed with the standard supplier tool chain, thus avoiding usage of specific tools to generate adapter layers with specific interfaces mechanisms.
One of the main objective of AUTOSAR as standard is to ensure portability of functions towards different platforms and thus ease integration of Software Components from different development partners. Therefore data descriptions are defined in an XML file, which can be processed by a tool during the software build phase to generate automatically the appropriate header and source code files for data declaration and definition.

When sharing code with a supplier, the alignment of standard used at OEM side with standard used by the integrating supplier allows an enhanced efficiency in the continuous integration process; thus necessary software adaptations can be automatized more easily.

Data consistency is a crucial element in the automotive embedded software development (especially when using pre-emptive and cooperative tasks in a single- or multicore context). Data specification information within standard files (MDX and AUTOSAR files) can be used to ensure the data consistency and thus these files can be directly be processed during the software build phase to take care on data consistency mechanism in the final software (hex-file). For MDX file, the so called tool MCOP (Message Copy OPtimizer) is used whereas for AUTOSAR files, the RTE generator ensures the data consistency mechanisms especially for Sender-Receiver implicit communication.
The configuration of MDX objects for a Simulink® model is performed using the MDX explorer (derived from standard Simulink® model explorer) and is part of the data dictionary to be shared between development partners and the configuration of AUTOSAR objects is performed in the workspace (to create AUTOSAR objects) as well as with the AUTOSAR Mapping Editor for Simulink® to prepare the

model for RTE configuration. Working with the same standard between OEM and supplier can reduce the effort of the model preparation at the supplier side for the code generation in case of Model Sharing Level 2 (or so-called Commissioned Development).

With the Model Sharing level 0, the MDX or AUTOSAR standards are supported by tools used at Bosch to generate the artefacts for Virtual Prototyping, done with INTECRIO or EVE [1] [2]. Some tests have been done with Software Sharing code which are not using MDX or AUTOSAR. A non-negligible effort had been spent to adapt the tool and scripts to the code and some information were missing and the way to extract is not as reliable as with an MDX or ARXML file.

4. The ASAM-MDX standard

The ASAM-MDX standard is supported by the tool chains at Bosch DGS-EC for years and the MDX target is part of the Simulink® tool chain. The name of the MDX files is in form of *_mdx.xml. The structure of the MDX files is organized into several parts:

- Data dictionary for the variables: it describes the implementation information such as the Datatype, the Compu Method, the Data Constraints, the Memory Location, the type of communication
- Data dictionary for the parameters: it describes the implementation information of each variable and parameters such as the Datatype, the Compu Method, the Data Constraints, the Memory Location, the Record Layout
- Description of the Compu Method
- Description of the Data Constraints
- The services: for each runnable, it describes the direction for each interface (READ, WRITE or READWRITE) and the type of access. The information provided in this part is very important for the tools which ensure the data consistency
- A list of internal variables, interfaces, parameters and runnable defined in the Software Component
- A list of interfaces and parameters which are exported by the Software Component
- A list of interfaces and parameters which are imported by the Software Component

The definition of the Compu Methods and the Data constraints can be done in self-containment or centrally defined.
In addition to the MDX files, a CDF file can be used to define the default values for the parameters.

The MDX and CDF files are generated by Simulink® with the AddOn MDX Exporter from MathWorks. It is necessary to create MDX elements in the Workspace such as *MDX.Variable* and *MDX.CalPrm*. Some information, which are not present in these Workspace elements like memory location or scope (import, local, export), are defined in the MDX Explorer. It is accessible from the Simulink® model and its content is embedded in the model.
A primary role of the MDX files is to generate automatically the headers files for the data declaration during the software build phase. The headers are enriched with other information. It avoids to generate them during the code generation phase. Hence it permits to have a flexible configuration depending on each project.
Another primary role is to be able to generate automatically an A2l file. The A2l is a standard file from ASAM association which contains the complete data description for the calibration tools like INCA. Without XML files like MDX file, it would be impossible to generate the A2l file with the c code and header file only since they do not contains the Compu Method description for example.

Describing all the data and the scope of each interface in an XML file for every Software Components offers many other advantages, especially with self-containment. During the software build phase, it is indeed possible to do a check if there are open interfaces. The software build can be interrupted in an earlier phase than without this functionality and a log file that provides a list of the issues is generated. It is also possible the check the data definition incoherencies for the interfaces: for example a data type difference between an imported interface and the exported interface.

With the new generation of microcontrollers with Multi-Core technology, the MDX standard provides further advantages for the integrating suppliers regarding the data consistency constraints. With Multi-Core microcontroller, the runnable in the Application Software part can be distributed over several cores. The MDX standard introduces the concept of import and export messages. The data consistency between inputs and outputs communicating between two different cores has to be ensured. In the Bosch DGS-EC software build tool chain, this is done with the tool MCOP. It is a tool included in the DGS-EC

software build tool chain and it uses the READ/WRITE access defined in the MDX files as input to determine which data has to be protected by doing a data consistency analysis. By using MCOP solution with MDX files, there is no need to implement data consistency features in the c code, since the consistency is implemented during the software build process. The consistency implementation is adapted to every software configuration (task distribution and functions scheduling).
It is a mechanism similar to the implicit communication with the AUTOSAR sender-receiver interfaces. The main difference remains in the fact that the implementation of the data protection with MCOP and MDX files is done in the object code and the ELF file with the VARED mechanism during the compilation phase. With AUTOSAR, the data protection is implemented during the RTE generation, in an earlier phase of the software build process.

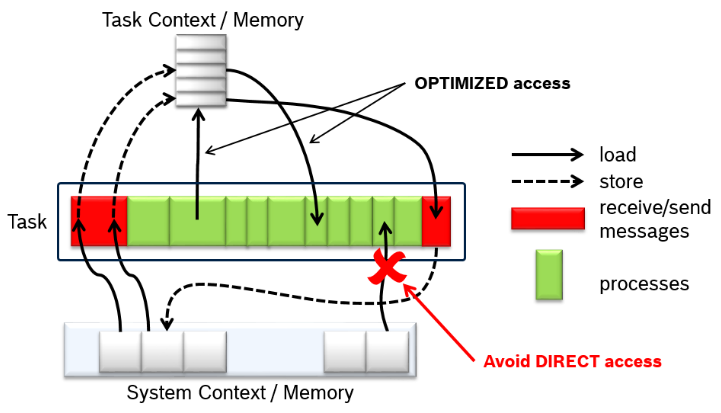
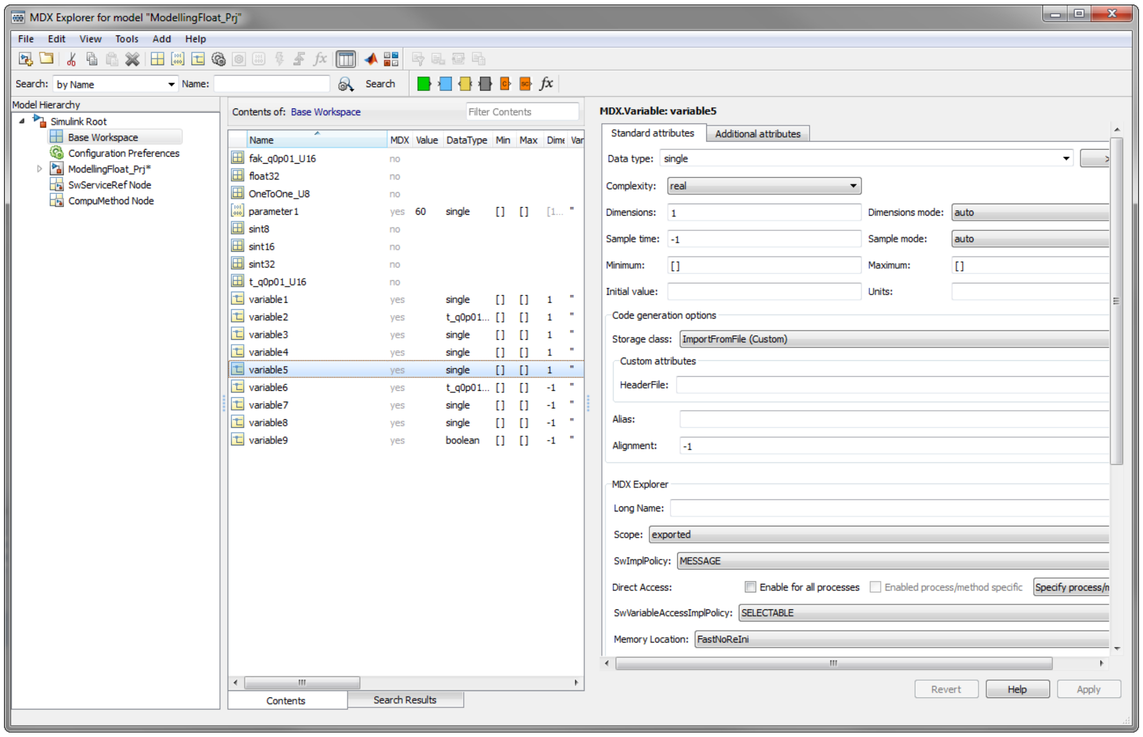


Figure 3 – Bosch MCOP concept

In a Model Sharing context, the support of the MDX standard is depending on the modelling tools and their configurations. With ASCET tool, the Bosch DGS-EC target includes a feature which permits the code generator to generate an XML file called PaVast which has a format and a semantic very similar to the MDX format. Both format are supported by the same tool chain during the software build process. If an OEM wants to exchange ASCET models with Bosch more easily, it is recommended to use a compatible target to avoid the use of a conversion tool.
With the Simulink® tool, the MDX files can be generated with the MDX Exporter from MathWorks. It can be either configured in the OEM Simulink® environment or it is included in the Bosch DGS-EC Add-On which offers a complete code generation environment compatible with the Bosch DGS-EC code development rules. This Add-On can be delivered to any OEM in the framework of a cooperation contract.
The cooperation between OEM and supplier when exchanging models can be eased when both development partners are using the same MDX standard for their Model Based Development tool chain.
It can be a blocker when exchanging library blocks for the code generation. Some library blocks, like filters or flip-flop, may include memory states. For example, the memory states can be described with the Workspace elements *MDX.Variable*. If the user of a Simulink® model who need this library does not have the MathWorks MDX Exporter, he shall do with a script a conversion of MDX Workspace elements into Simulink® elements in order to be able to run a simulation.
With a Model Sharing Level 1 or 2, if he OEM delivers Simulink® models without any MDX support, the MDX configuration can be added by the supplier. The adaptation is done in two steps. The first step is dealing with a conversion of the OEM data dictionary into MDX elements in the Workspace. The conversion task can be eased if the OEM is able to provide a data dictionary in MDX format. It can be more easily converted into MDX elements since the XML format can be processed more easily than Excel format and is less sensitive to issues with bad fields' content with bad characters.

The second step is dealing with information which are not present in these Workspace elements like memory location or scope (import, local, export). They shall be defined in the MDX Explorer which is accessible from the Simulink® model and these data are embedded in the model. The filling of the MDX Explorer can be done more easily if the OEM is able to provide a data dictionary in MDX format containing the useful information like the scope (import, local, export). The information can also be completed with Matlab scripts through specific API.


MDX Explorer for model "ModellingFloat_Prj"

Model Hierarchy: Simulink Root; Base Workspace; Configuration Preferences; ModellingFloat_Prj*; SwServiceRef Node; CompuMethod Node

Contents of: Base Workspace

Name	MDX	Value	DataType	Min	Max	Dime	Var
fak_q0p01_U16	no						
float32	no						
OneToOne_U8	no						
parameter1	yes	60	single	[]	[]	[1...	"
sint8	no						
sint16	no						
sint32	no						
t_q0p01_U16	no						
variable1	yes		single	[]	[]	1	"
variable2	yes		t_q0p01...	[]	[]	1	"
variable3	yes		single	[]	[]	1	"
variable4	yes		single	[]	[]	1	"
variable5	yes		single	[]	[]	1	"
variable6	yes		t_q0p01...	[]	[]	-1	"
variable7	yes		single	[]	[]	-1	"
variable8	yes		single	[]	[]	-1	"
variable9	yes		boolean	[]	[]	-1	"

MDX.Variable: variable5

Standard attributes | Additional attributes

Data type: single
Complexity: real
Dimensions: 1 Dimensions mode: auto
Sample time: -1 Sample mode: auto
Minimum: [] Maximum: []
Initial value: Units:

Code generation options
Storage class: ImportFromFile (Custom)
Custom attributes
HeaderFile:
Alias:
Alignment: -1

MDX Explorer
Long Name:
Scope: exported
SwImplPolicy: MESSAGE
Direct Access: Enable for all processes Enabled process/method specific
SwVariableAccessImplPolicy: SELECTABLE
Memory Location: FastNoReIni

Figure 4 – Simulink MDX Exporter

With the use of XML files for the support of a standard like MDX, it is strongly recommended to use an automatic code generator. The MDX files contain in general hundreds of lines of code with tags.

5. The AUTOSAR standard

Bosch DGS-EC is working on an efficient use of the AUTOSAR standard in a scope of development for series. It is a more complete standard than the MDX standard since the Basic Software is standardized and clearly decoupled from the Application Software. It describes the concept of Virtual Functional Bus which is supported by the Run Time Environment (or RTE). In the same way as the MDX standard, the data are described in a XML file, it is called ARXML (AUTOSAR XML). It offers a much more complete semantic and is based on AR Packages and AUTOSAR element references. Contrary to the MDX standard, the infrastructure interfaces such as diagnostic, NVM, DCM interface are supported as standardized Basic Software interfaces. It is a main advantage in comparison with MDX since the BSW interfaces can be implemented in the models and described in the ARXML files. With Software Sharing context, it avoids using implementing Sender-Receiver interfaces in the Software Components and developing Customer Interface Layers to adapt these Sender-Receiver interfaces to the integrating supplier BSW specific interfaces.
Another advantage of the AUTOSAR standard is dealing with standardized AUTOSAR service libraries. These service libraries have standardized names and argument names. Their algorithms are also precisely described so that they can be implemented by any integrating supplier. AUTOSAR service routines do not use AUTOSAR mechanisms: they can be used for developments with MDX standard as well as AUTOSAR.
With the Simulink® tool chain, Bosch DGS-EC is able to deliver Simulink® library blocks supporting the AUTOSAR service libraries, since they are not all available with the native MathWorks blocks.

AUTOSAR is supported by many tools from the market, this is an important advantage since the in-house development of specific software development tools can be avoided. If an OEM is working with AUTOSAR, then there is no need for its integrating suppliers to develop specific tool. It would be very costly if every OEM and supplier had their own standard. AUTOSAR has changed the landscape of the manufacturer - supplier relationship. With this standard, it can much easier than ever to integrate the code of an OEM or a Third Party into an integrating supplier software. It is a mean to ease the Software Sharing and open new cooperation frameworks. The structure of the ARXML file is however not fixed as for MDX: the structure ordering is depending on the AR-Package structure and AR-Packages name. The AR-Packages can be spread over several ARXML files and a Software Component can be described by several ARXML files, for example one ARXML file for each AR-Package. The elements properties are not defined only by short name, but by reference to the AR-Packages.

The main automotive modeling tools ASCET, TargetLink® or Simulink® are supporting AUTOSAR. With ASCET, the concept of AUTOSAR Model has been introduced in the tool. It is very similar to the non-AUTOSAR models, especially regarding the modelling of the functionalities. From an architecture point of view, an atomic Software Component is represented by one ASCET module. A composition can be represented in the ASCET Project by an assembly of ASCET modules.

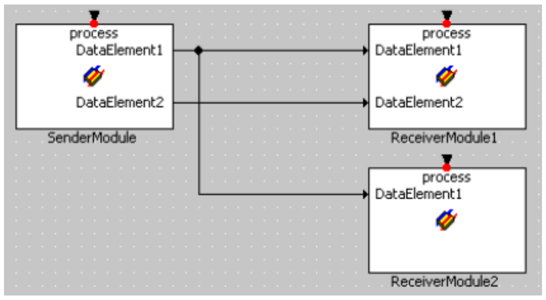


Figure 5 – Composition in ASCET

Instead of Import / export message blocks, there is a specific block available for the Sender/Receiver interfaces and the type of communication (explicit VS implicit) can be selected.

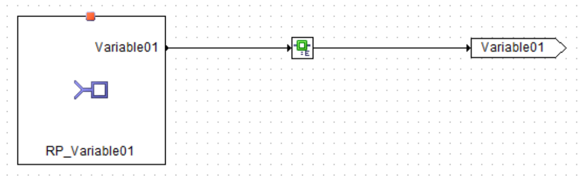
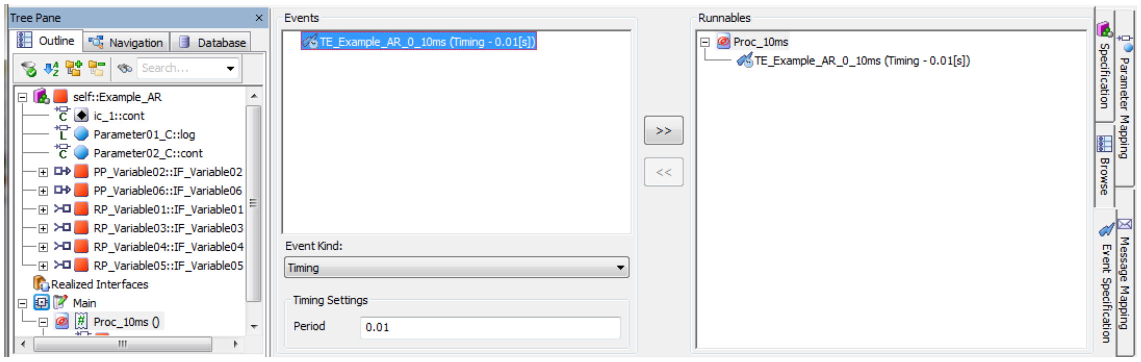


Figure 6 – AUTOSAR ports modelling in ASCET

An additional tab has been introduced to configure the runnable to events mapping. The naming convention for the AR-Package and many AUTOSAR elements are configured in an XML configuration file.



TE_Example_AR_0_10ms (Timing - 0.01[s])

Event Kind: Timing

Timing Settings

Period 0.01

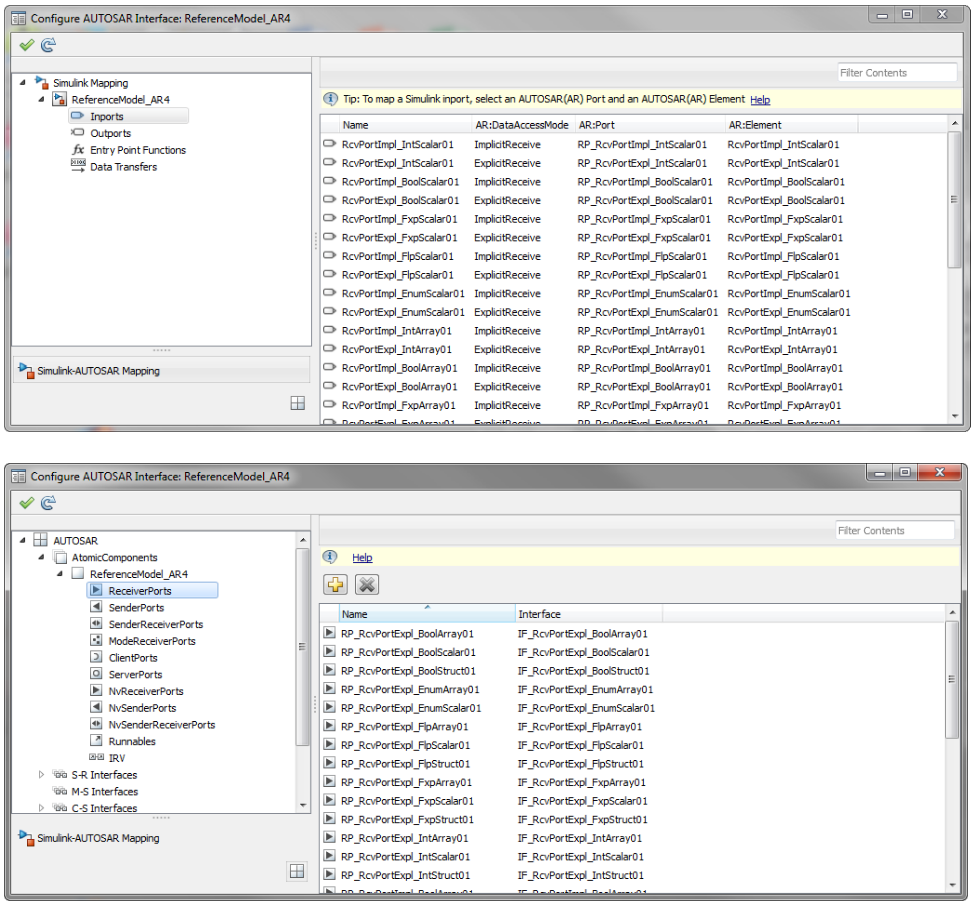
Proc_10ms
TE_Example_AR_0_10ms (Timing - 0.01[s])

Figure 7 – ASCET runnable to event mapping

For the development of AUTOSAR Software Components, a cooperation between MathWorks and Bosch has been established within the framework of a strategic partnership in order to support AUTOSAR with Simulink® in an efficient way. At first glance, a Simulink® model for AUTOSAR looks like a model for MDX. There are three main differences. The first one affects the Workspace elements: it is necessary to create AUTOSAR elements such as *AUTOSAR.Signal* and *AUTOSAR.Parameter*. ARXML files can be imported through an API to load for example the Internal Behavior or central elements for the central definition of AUTOSAR elements such as base types or Compu Method.

The second difference is dealing with the AUTOSAR Mapping Editor which contains two sheets: the ports, the interfaces (Sender/Receiver, Client/Server), the IRV and the runnables are defined in the first sheet. Their mappings to the models inports, outports and the function call is ensured in the second sheet. The content of the AUTOSAR Mapping Editor can be filled automatically with a script thanks to

APIs. The script shall be configured to fit to the naming convention applied in the environment development. For example, a Receive Port could have a prefix "R_" or "RP_". This automation is possible if the ports, interfaces and data elements share the same name root. This is a rule established at Bosch DGS-EC and it allows to do also an automation of the AUTOSAR composition, called "Autocomposition".



Configure AUTOSAR Interface: ReferenceModel_AR4

- Simulink Mapping
  - ReferenceModel_AR4
    - Inports
    - Outports
    - Entry Point Functions
    - Data Transfers

Simulink-AUTOSAR Mapping

Filter Contents

Tip: To map a Simulink inport, select an AUTOSAR(AR) Port and an AUTOSAR(AR) Element Help

Name	AR:DataAccessMode	AR:Port	AR:Element
RcvPortImpl_IntScalar01	ImplicitReceive	RP_RcvPortImpl_IntScalar01	RcvPortImpl_IntScalar01
RcvPortExpl_IntScalar01	ExplicitReceive	RP_RcvPortExpl_IntScalar01	RcvPortExpl_IntScalar01
RcvPortImpl_BoolScalar01	ImplicitReceive	RP_RcvPortImpl_BoolScalar01	RcvPortImpl_BoolScalar01
RcvPortExpl_BoolScalar01	ExplicitReceive	RP_RcvPortExpl_BoolScalar01	RcvPortExpl_BoolScalar01
RcvPortImpl_FxpScalar01	ImplicitReceive	RP_RcvPortImpl_FxpScalar01	RcvPortImpl_FxpScalar01
RcvPortExpl_FxpScalar01	ExplicitReceive	RP_RcvPortExpl_FxpScalar01	RcvPortExpl_FxpScalar01
RcvPortImpl_FlpScalar01	ImplicitReceive	RP_RcvPortImpl_FlpScalar01	RcvPortImpl_FlpScalar01
RcvPortExpl_FlpScalar01	ExplicitReceive	RP_RcvPortExpl_FlpScalar01	RcvPortExpl_FlpScalar01
RcvPortImpl_EnumScalar01	ImplicitReceive	RP_RcvPortImpl_EnumScalar01	RcvPortImpl_EnumScalar01
RcvPortExpl_EnumScalar01	ExplicitReceive	RP_RcvPortExpl_EnumScalar01	RcvPortExpl_EnumScalar01
RcvPortImpl_IntArray01	ImplicitReceive	RP_RcvPortImpl_IntArray01	RcvPortImpl_IntArray01
RcvPortExpl_IntArray01	ExplicitReceive	RP_RcvPortExpl_IntArray01	RcvPortExpl_IntArray01
RcvPortImpl_BoolArray01	ImplicitReceive	RP_RcvPortImpl_BoolArray01	RcvPortImpl_BoolArray01
RcvPortExpl_BoolArray01	ExplicitReceive	RP_RcvPortExpl_BoolArray01	RcvPortExpl_BoolArray01
RcvPortImpl_FxpArray01	ImplicitReceive	RP_RcvPortImpl_FxpArray01	RcvPortImpl_FxpArray01

Configure AUTOSAR Interface: ReferenceModel_AR4

- AUTOSAR
  - AtomicComponents
    - ReferenceModel_AR4
      - ReceiverPorts
      - SenderPorts
      - SenderReceiverPorts
      - ModeReceiverPorts
      - ClientPorts
      - ServerPorts
      - NvReceiverPorts
      - NvSenderPorts
      - NvSenderReceiverPorts
      - Runnables
      - IRV
  - S-R Interfaces
  - M-S Interfaces
  - C-S Interfaces

Simulink-AUTOSAR Mapping

Filter Contents

Help

Name	Interface
RP_RcvPortExpl_BoolArray01	IF_RcvPortExpl_BoolArray01
RP_RcvPortExpl_BoolScalar01	IF_RcvPortExpl_BoolScalar01
RP_RcvPortExpl_BoolStruct01	IF_RcvPortExpl_BoolStruct01
RP_RcvPortExpl_EnumArray01	IF_RcvPortExpl_EnumArray01
RP_RcvPortExpl_EnumScalar01	IF_RcvPortExpl_EnumScalar01
RP_RcvPortExpl_FlpArray01	IF_RcvPortExpl_FlpArray01
RP_RcvPortExpl_FlpScalar01	IF_RcvPortExpl_FlpScalar01
RP_RcvPortExpl_FlpStruct01	IF_RcvPortExpl_FlpStruct01
RP_RcvPortExpl_FxpArray01	IF_RcvPortExpl_FxpArray01
RP_RcvPortExpl_FxpScalar01	IF_RcvPortExpl_FxpScalar01
RP_RcvPortExpl_FxpStruct01	IF_RcvPortExpl_FxpStruct01
RP_RcvPortExpl_IntArray01	IF_RcvPortExpl_IntArray01
RP_RcvPortExpl_IntScalar01	IF_RcvPortExpl_IntScalar01
RP_RcvPortExpl_IntStruct01	IF_RcvPortExpl_IntStruct01

Figure 8 – Simulink AUTOSAR Mapping Editor

The third difference is dealing with the accesses to the infrastructure: there are described with Client-Server interfaces and many of them have standardized names. "Simulink® Function" and "Simulink® Caller" blocks can be used.

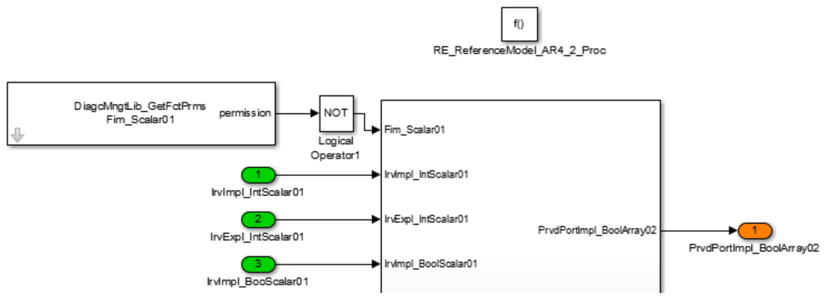


Figure 9 – AUTOSAR FiM interface modelling in Simulink

The software build process has to be adapted to an AUTOSAR project: in comparison with non-AUTOSAR project, a tool called RTE Generator shall be invoked. It generates c and h files from the ARXML abstraction layer. It generate the RTE which ensure the communication. In most of the projects; there is a mix-mode with a part of the software in MDX or any other specific OEM standard. The implementation of Customer Interface Layers is required and is a drawback to a smooth transition to

AUTOSAR in comparison to a “big bang” introduction. It requires then additional development effort and additional RAM consumption. There is also an impact on the OS scheduler: a pure AUTOSAR scheduler can not be implemented. The legacy OS scheduler references RTE task container tasks which are described in a configuration ARXML.

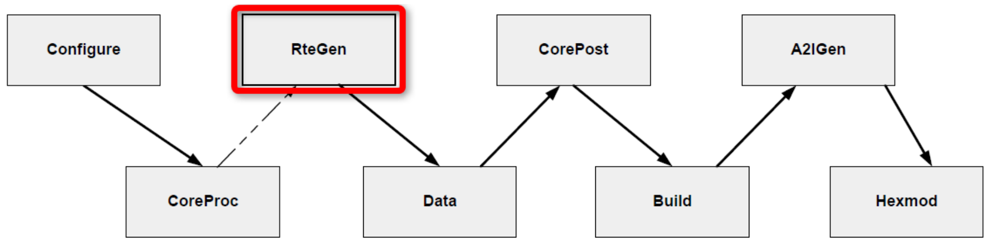


Figure 10 – Bosch Software build Process

The implicit Sender-Receiver communication is recommended to ensure the data consistency for the communication between two different cores in a Multi-Core microcontroller or between two different tasks with a pre-emptive Operating System (OS). Buffers are created during the RTE generation in the c and h files. This is done indeed before the compilation phase and it does not require specific tools as for MDX standard, since the protection mechanisms are described in the standard and supported by the RTE Generators.

In a Model Sharing context, the tools ASCET, TargetLink® or Simulink® offer the possibility to develop AUTOSAR Software Components without spending so much time on the target configuration. With ASCET tool, an AUTOSAR target configuration is available and the AUTOSAR model can be quite easily exchanged between OEM and suppliers. It is however very important to keep in mind that some AUTOSAR features availability is depending on the AUTOSAR release. All the partners in a cooperation framework shall agree on the supported release before starting the development to avoid model incompatibilities. In the same way, it is very important for an OEM or a third party to agree on the AUTOSAR Release supported by the RTE Generator. They have to be sure that the required AUTOSAR features in the models are supported by the RTE Generation version in use for the project.
With the Simulink® tool, the same precautions have to be taken regarding the AUTOSAR release compatibilities. As an example, the PRPortPrototype feature has been introduced AUTOSAR 4.1.1. If the development partners are working with different Simulink® versions, this feature may not be available in an older version and there would be an issue in the AUTOSAR Mapping Editor Content. With the same example, this feature could be generated in the ARXML but it is not sure whether the RTE Generator version in use at the integrating supplier side would support this feature. Developing an AUTOSAR Software Component has an impact on the modelling patterns. For example, multi-triggering on runnable level is not possible, since one runnable shall be mapped with only one event. The standardization of the BSW interfaces is a great enabler for Model Sharing. An obvious example are the DEM / FIM diagnostic interfaces. In comparison with MDX, every development partners can use the same service access and there is no need to develop customer specific services on the integrating supplier side. As a drawback it reduce the possibilities to bring innovations in these services and the OEM and suppliers have less levers to differentiate their technology from their competitors.

Shared software development for electronic control units becoming more and more important. In order to cope with the complexity of the Software development for Engine Control, a strong and highly flexible Software architecture is required. The answer of Bosch to this challenge is the creation of the Software architecture VeMotionSAR^{TM} that strengthens the cooperation with the OEMs and that has been published in Internet (http://www.bosch-vemotionsar.com). One of the main benefits of this architecture is the modularity of the function description and the definition of the interfaces. It permits to describe individually the various specifications of the corresponding development partners’ functionalities in a highly modular architecture domain according to their resources / tools.

6. Migration from legacy standard to MDX

MDX standard is mature and in use at Bosch DGS-EC for years for series project. With the new Multi-Core microcontrollers and the need of data consistency mechanisms, it can be a good alternative for the integrating suppliers who are using specific standards which are not supporting Multi-Core. It is therefore interesting to analyze the impacts and the constraints on Model Sharing cooperation’s when doing a migration to the MDX standard. AUTOSAR is a promising standard in the automotive embedded software and more and more control units have an AUTOSAR RTE with a partly or complete AUTOSAR Application Software. The description of migrations uses cases are focused on Simulink® tool in this chapter.

For the conversion of Simulink® models to MDX, the main challenges are oriented to the migration of the Workspace elements and the configuration of the MDX Explorer. The Workspace elements can be easily converted to MDX elements with Matlab scripts. Some information contained in the Workspace elements could be lost if the original models are using Custom Storage Class. Depending on the features, they can either be taken into account in the MDX Explorer or may lead to some changes in the model. For example the attribute to know if a variable shall be stored in the Non Volatile Memory can be set in a Custom Storage Class. During the migration, this attribute shall be taken into account in the MDX Explorer by setting the memory location to a Non Volatile Memory location. This example demonstrates that the MDX Explorer content shall be updated at the same time as the Workspace elements. Since all the MDX Explorer attribute can be accessed with APIs, a m file for the MDX Explorer configuration can be created during the migration and running it will fill the MDX Explorer content. This is the solution chosen at Bosch for converter scripts.
The migration to the MDX standard may have also an impact on the modeling pattern in a Simulink® model. A frequent issue is faced for inport which may uses the value of the outports in the same runnable. It is not possible to have the same name for the inport and the outport, the inport shall be removed and the signal shall be replaced by a Unit Delay block connected to the outport. It is more complicated to handle it if the same outport is also initialized in a separate initialization runnable. For this case, a specific UnitDelay block, which includes a memory state value defined in the Workspace as a *MDX.Variable,* has been created at Bosch DGS-EC. Another impact on the modelling patterns is dealing with the Multi-Core constraints: it is not possible to keep the multi-triggering on runnable level. The functionality need to be refactored or the runnable content needs to be duplicated. The duplication may be complicated in case of state machine or any functions containing state values.

7. Migration from legacy standard or MDX to AUTOSAR

For the conversion of Simulink® models from legacy or MDX to AUTOSAR, the main challenges are oriented to the migration of the Workspace elements and the configuration of the AUTOSAR Mapping Editor. The most important challenge for developments with AUTOSAR is to be as efficient as the standard it is supposed to replace in a defined tool chain. For ECU Software development with its complexity, it is reasonable to consider that it would not lead to short-term cost reduction. However, it could avoid spending more efforts in long-term by reducing the development and maintenance effort of specific tools n case an OEM or a supplier is using its own standards . A migration to AUTOSAR makes sense if the tool chain is ready for efficient development with AUTOSAR from an economic point of view.

The experience shows that it is necessary to write some scripts to reduce the manual effort and the risk of error. The advantage of Model Based Development lies in the fact that the AUTOSAR adaptation can be done directly in the model used for the code generation. Prior to the migration task, many clarifications with the OEM are necessary such as:

- Architecture: one model = one atomic SW-C, or one model = one Runnable ?
- AR-Package structure
- Naming conventions for the AR elements short names
- Restriction on the naming rules for the ports, interface and data elements to allow an autocomposition

The Workspace elements can be converted into AUTOSAR elements with Matlab scripts. At Bosch DGS-EC, the measurement points are defined as Per Instance Memory (PIM) whereas they are *Simulink.Signal* or *MDX.Variable* with other targets where the inputs and outputs are also *Simulink.Signal* or *MDX.Variable*. The script shall detect the inputs and the outputs in the models in order to identify which *Simulink.Signal* or *MDX.Variable* shall be converted into *AUTOSAR.Signal* and the other into PIM. The AUTOSAR Mapping Editor can filled automatically with a script.
There are also impacts on the modelling patterns:

- The multi-triggering on runnable level is not compatible with AUTOSAR: with a Simulink® model, it is not possible to have a multi-triggering on the Function Call subsystems representing the runnable since one runnable shall be mapped to ne event.
- The messages exchanged between runnable shall be converted as Inter Runnable Variables (IRV).

- With Simulink, the measurement points are defined in MDX as global variable with a resolve signal to Workspace: in AUTOSAR they shall be defined as static memory or Per Instance Memory (PIM).
- At Bosch DGS-EC, many legacy services can not be mapped one to one to AUTOSAR standardized client-server interfaces. In case of software sharing, these interfaces shall be replaced by Sender-Receiver interface and the integrating supplier shall map them to its equivalent services via an adapter.
- the physical system constants are not supported by AUTOSAR, a work-around is necessary

In order to prepare a seamless migration to AUTOSAR for mid or long term, some measures can be taken in the modelling pattern rules for the legacy models to ensure a compatibility with AUTOSAR. The modeling pattern shall define so that a model could converted automatically into AUTOSAR without any rework.

- For the interpolation, it necessary to have blocks which support MDX and AUTOSAR at the same time. The Simulink® look-up table will support soon both MDX and AUTOSAR standard with AUTOSAR R4.x libraries Ifx and Ifl.
- For the Dem and FiM interface in AUTOSAR, if similar services are available in legacy environment, it could be possible to create Simulink® blocks with some an automatic configuration inside, depending on the choice between the legacy or AUTOSAR target.
- Legacy services which are not identical with the AUTOSAR similar service require to define modeling pattern adaptations.

Summary

Model Sharing is currently applied at Bosch for several serial projects and is leading to clear improvements in development efficiency. Use of Standards (e.g. ASAM (MDX, CDF …) AUTOSAR (service libraries, methodology …) as well as standardized interfaces specifications (e.g. as published with AUTOSAR R4.x) can significantly ease exchange of models.

MDX standard is supported by the code generation tool chains for years at Bosch and permits to perform interfaces coherency check during the Software Build process as well as to provide information to build the A2l file. With the Multi-Core microcontrollers, the MDX files are used by the tool MCOP to ensure the data consistency.

The AUTOSAR standard clearly defined a separation between the Basic Software and the Application Software and provides standardized interfaces with standardized mechanism. In comparison with MDX, the infrastructure interfaces with the Basic Software are standardized which is a lever to improve the model exchanges. For engine control unit software, the Process, Method and Tool for AUTOSAR developments are not yet as efficient as with MDX development. Bosch is working actively with the tool editors to reduce the gaps. On one side, Tier 1 need to know if the OEM want to base their future development with the AUTOSAR standard. On the other side, OEM want to know when the Tier 1 will be able to support AUTOSAR in an efficient way. The approach at Bosch has been to edit internal AUTOSAR coding guidelines. The current tasks are focused on the identification and the reduction of gaps between those guidelines and what is supported by the tools.

It is possible to migrate a legacy Simulink model to MDX or AUTOSAR. Many migration tasks can be automatized with scripts. The modelling patterns are also impacted and if the migration to AUTOSAR is a long-term strategy, it is recommended to take into account this modelling constraints in the legacy models in short term.

Bibliography:

1. Dr. M. Tanimou, Dr. M. Münzenmay, K. Zimmermann, Robert Bosch GmbH; Dr. B. Lumpp, Dr. E. Trapel, E. Bouillon, Dr. M. McMackin, MAN Truck & Bus AG: Software-in-the-Loop durch Co-Simulation von EDC-Modell und GT-Modell; Mainz, 14th MTZ-Conference 2014-01-0189 - virtual powertrain creation, 2013
2. Lumpp, B., Tanimou, M., McMackin, M., Bouillon, E. et al., "Desktop Simulation and Calibration of Diesel Engine ECU Software using Software-in-the-Loop Methodology," SAE Technical Paper 2014-01-0189, 2014, doi:10.4271/2014-01-0189
3. S. Louvet, Robert Bosch (France) SAS, Dr. U. Niebling, Dr. M. Tanimou, Robert Bosch GmbH : Model Sharing to leverage customer cooperation in the ECU software development; Toulouse, ERTS 2014-Conference, 2014

A Model-driven and Tool-integration Framework for Whole Vehicle Co-simulation Environments

Jinzhi Lu, Dejiu Chen, Martin Törngren, Jad El-Khoury, Frédéric Loiret
School of Industrial Engineering and Management
KTH Royal Institute of Technology, Stockholm 100 44
jinzhl@kth.se

***Abstract*— Throughout the design of automotive vehicle systems, modeling and simulation technologies have been widely used for supporting their conceptualization and evaluation. Due to the increasing complexity of such systems, the overall quality management and design process optimization are becoming more important. This in particular brings the necessity of integrating various domain-specific physical models that are traditionally based on different formalisms and isolated tools. In this paper, we present the initial concepts towards a model-driven and tool-integration framework with automated managed simulation services in the system development. We exploit EAST-ADL and some other existing state-of-the-art modeling technologies as the reference frameworks for a formal system description, with the content including requirements, design solutions, extra-functional constraints, and verification and validation cases. Given such a formal specification, dedicated co-simulation services will be developed to provide the support for automated configuration and execution of simulation tools.**

Keywords—Development, Simulation, System Design, Model-driven, Tool-integration

I. Introduction

Automotive vehicle systems have evolved continuously over the past decades with increasing functional and technical complexity. The development involves complex interactions of designers and stakeholders, [1]. Typically, an automotive vehicle can be divided into different parts with associated engineering tasks allocated to different work groups or companies. These groups often use specific domain-specific simulator tools to assist the specification, analysis and synthesis tasks.

For a whole vehicle, the overall system behaviors and other properties depend on the characteristics of its subsystems and components being composed. It is therefore important for system developers to conduct early system integration and thereby to understand the system behaviors and analyze system performance in the initial design phases.

In current industrial practices, different CAD and CAE tools have been used for the development of vehicle subsystems and components, such as for requirement analysis, high level design, detailed level design, implementation, testing, etc. Since the subsystem designers often use their own tools to build models, it is a challenge for the system developers to integrate those detailed models and to predict the whole system performance. This calls for a complete integration framework for different models and tools while taking the process management perspective into consideration.

This paper describes an initial version of, and work towards establishing, a model-driven and tool-integration design framework for whole vehicle systems in regard to the methodology and technical roadmap. In such a framework, a formal system description is provided to share information among various stakeholders, including project managers, system engineers, modeler, system designers and simulation testers. The content of system information being described and integrated includes requirements, function analysis, system design, parameter settings, interface contract of physical system models, and tool specific information for co-simulation. The framework integrates subsystem models in order to predict the whole vehicle performance. Also, parameters related to optimized simulation behaviors can be added.

The paper has five main sections. Section 2 provides a brief introduction of simulation technologies and system modeling methods. Section 3 presents a problem analysis of an auto-breaking system. Section 4 illustrates a model-driven and tool-integration framework we provided. Section 5 demonstrates a co-simulation test case of a vehicle auto-breaking system. Section 6 and 7 discusses how Open Services for Lifecycle Collaboration (OSLC) can be integrated into our framework and future work.

II. State of the Art

Nowadays, computation design and simulation technology can help to improve design efficiency and decrease the time consumption and R&D cost [2]. In the vehicle conception design phase, model-based design is very important to predict the system performance in advance. Designers have used many types of modeling technologies and tools to finish their design jobs. For example, CAE and CAD tools, such as Solidworks, Catia, AMESim, Simplorer, Matlab\Simulink, are very widely used in the vehicle industries.

However, there are still various challenges in the whole life cycle of vehicle design. Firstly, different CAD and CAE tools can help designers to build models for designing these subsystems or components to satisfy design requirements, however the whole system performance is very different (and more difficult) to predict, because the subsystem designers are often from other groups or companies and use different tools to build models.

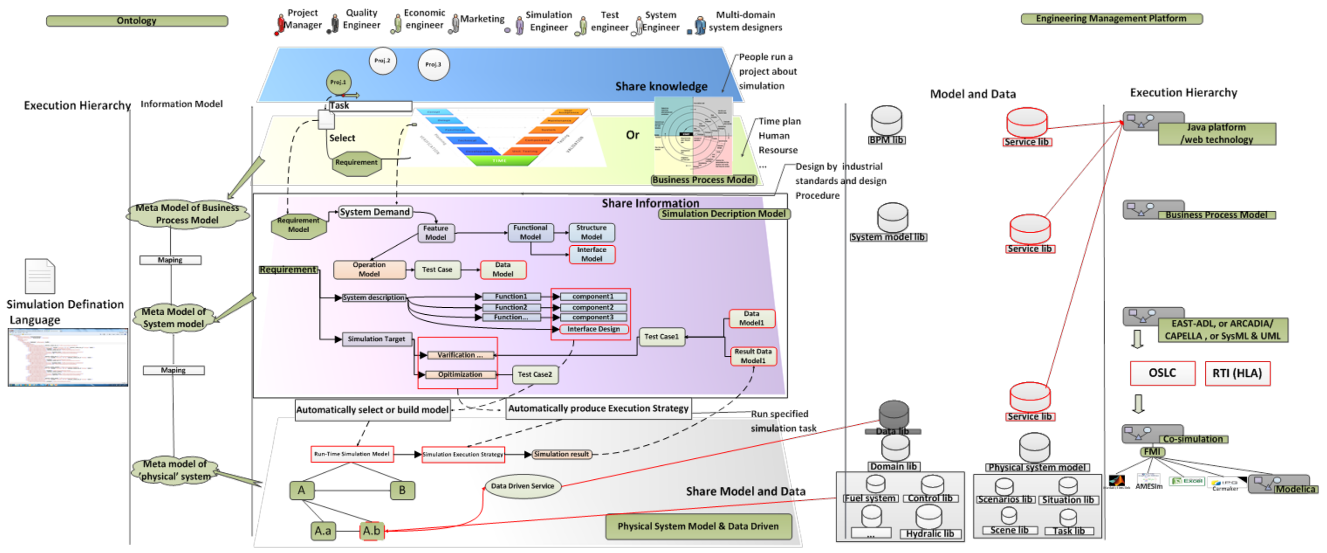


FIGURE 1 MODEL-DRIVEN AND TOOL-INTEGRATION FRAMEWORK FOR WHOLE VEHICLE SIMULATION

Sometimes because of the intellectual property, they even cannot provide the original models. That's why it's hard for integrated system department to integrate the detailed models to predict the whole system performance.

Secondly, even though various PDM and PLM systems can help companies to manage information during vehicle design, such tool-based information management has a restricted scope. For example, if some subsystems are designed by different companies, it is no possible to push all the subsystem suppliers to use the same tool.

Thirdly, nowadays documents still play a very important role in communication and information description, but it's difficult to manage the changes in documents and releases of a huge document may delay the other parts of development. Consequently, a framework with formalized and graphical conception models for model-driven and tool-integration of co-simulation is needed. Several technologies that may be used for such a framework are now briefly surveyed.

A. *Multi-domain Simulation Technologies*

In addition to commercial simulation platforms, several languages and standards are provided for multi-domain system simulation and analysis.

1) Modelica

Modelica language is a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems [3]. Nowadays, Modelica language is widely used in automobile industries.

2) Co-simulation

Co-simulation is a technology which can solve the multi-physics model integration. It represents a particular case of simulation scenarios in which there are at least two simulators to solve coupled algebraic equations and exchange the data with each other during simulation [2].The High-Level Architecture (HLA) is a technology for developing distributed simulation developed by the Simulation Interoperability Standards Organization (SISO) [4]. The Functional Mock-up Interface (FMI) standard was initiated through the Modelisar project. FMI is designed for commercial simulators to transform their models to a normative form [5]. The HLA is powerful in mastering the whole co-simulation process and control the data flow. FMI is better for interface design for simulator tools.

B. *System Modeling Approach*

From the perspective of system engineering, physical system models are not the unique concern. Though a lot of complex and completed model libraries and model management platforms already exist, it's also different for system designers in each layer to understand the whole physical system model and share the model information with each other except for documents or reports. This leads to a need for an information model for information exchange and function descriptions. In this part, several system modeling languages, (e.g., SysML, EAST-ADL, etc.) are investigated.

1) SysML

SysML is a general-purpose modeling language for systems engineering with a subset of UML2 and additional extensions to satisfy the demands of the language description. SysML provides constructs for modeling systems engineering problems including requirements, structure, behavior, allocations, and constraints [6].

2) EAST-ADL

EAST-ADL is a language to describe automotive electrical and electronic systems; it relates closely to SysML and can be seen as a domain specific tailoring for automotive systems. It enables to capture detailed information for documentation, design, analysis and synthesis from the top level characteristics to tasks and interface &communication framework [7].

3) ARCADIA

Arcadia / Capella is a field-proven domain specific modeling solution for system engineering. It has an open source framework to define operation, system, logic and physical system [8].

III. Problem Analysis

We provide a test case to show how co-simulation can be used in autobreaking system design. The verification purpose is to test if the new autobreaking controller can satisfy the '3-second rule', which is a measurement on the time interval for the vehicles to pass the same fixed point on road. If a vehicle reaches such a point within 3 seconds after its foregoing vehicle, then the following distance is too short. For icy or snow-covered road, the corresponding interval can be 7~8 sec.

As Figure 2 shows, we define the Green zones, Red zones and Black zones for the autobreaking controller. X1 and X2 is the position in X direction of vehicle1's and vehicle2's geometric center. If the distance between V1 and V2 is not less than v1/1.2, vehicle2 need accelerate. When it's in red zones, vehicle 2 need decelerate. When the distance between V1 and V2 is less than 1.2m, these two vehicles crashed.

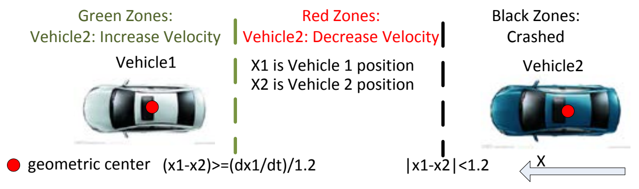


Figure 2 Checking Zones for autobreaking Strategy

In this case, Carmaker, MWorks and Simulink were used for subsystem modeling and the whole system model is integrated in Simulink which represents the co-simulation process. As shown in Figure 3, the controller model was built by Modelica language in MWorks and was exported to a FMU. Then S-function for co-simulation with FMU and interface library for Carmaker have been used to connect with these two subsystem models with Simulink.

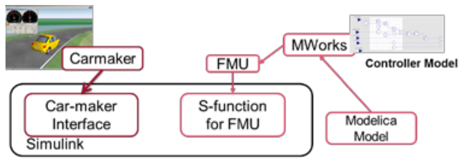


Figure 3 Tool-chain during Co-Simulation

When the co-simulation is running, Simulink Model, S-function for FMU interface, co-simulation environment setting should be set by hand. And when the version of FMU or Carmaker model is changed, the simulation result should be dependence to the changing. That's really difficult to manage the models and hard to configure the tool setting each time by hand. And simulation technology is used in whole life cycle. In [8], for example, simulation was used in different phases of 'V' model. So simulation information integration with different phase is also a big challenge for current environment without any tool or platform support.

IV. Model-driven And Tool-integration Framework

To solve the former challenge, we design a framework as shown in Figure 1. This framework is inspired by SPIT (Social layer, Process layer, Information layer and Technical layer) [9]. This framework is designed for tool-integration, data-integration, MBSE technical integration and web deployment for the whole life cycle in product design.

A. *Social layer*

In the social layer, the social models of people's behaviors, organizations, management views, social views and culture for MBSE (Model-based System Engineering) technology are created and describe relative engineering activity in the whole life cycle. From our research, we take an example of MBSE transitioning model as a social model in this layer to describe MBSE maturity level.

This MBSE maturity level model describes enterprises or research teams from the traditional document-based system engineering approach to MBSE approach. In the technical aspect, we separately define transitioning of software and multi-domain system design. As Table 1, we define transitioning level of software and multi-domain system. And we will have an investigation of different effect factors including cultural aspect, stakeholder aspect and transitioning aspect.

Table 1 Maturity Level of MBSE

Document-based Design	Software	Multi-domain system
Transitioning	Component Design	CAD-based Design
	Model-based Design	Simulation-based design and analysis
		Model-integrated Engineering and analysis
MBSE	Model-driven	Model-based development

B. *Business process layer (Process layer)*

In this layer, based on different stakeholders' requirements, domain or industrial standards and simulation targets, project managers or team leaders can build different kinds of Business process model (BPM) to describe the process of projects as Figure 6. For the distributed simulation as an example, the IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) is a standard to define engineering activities for the Process of simulation. We can see, in Figure 8,the BPMN can be used to describe the process model in DESSP.

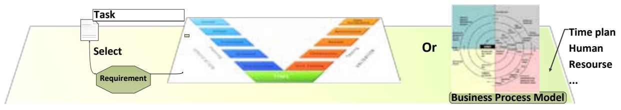


FIGURE 4 PROCESS LAYER

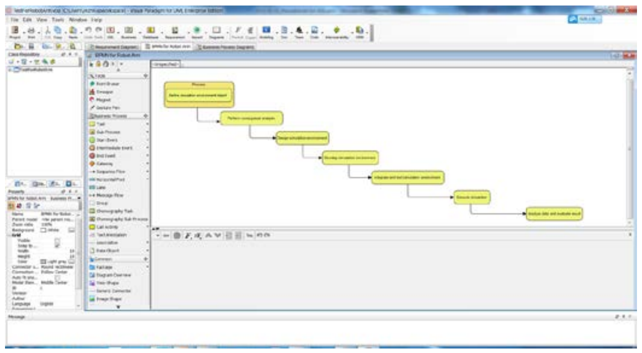


FIGURE 5 BPMN DESCRIBING DSEEP

C. *Simulation description model layer (Information layer)*

System designers can use system models to describe simulation requirement, physical system feature, model function, model structure, data model, simulation behavior model including verification model, optimization model, and so on as shown in Figure 7.

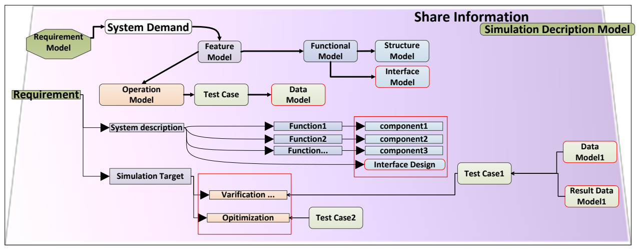


FIGURE 6 SIMULATION DESCRIPTION MODEL

In Figure 7, System model can describe simulation includes requirement diagrams, feature diagrams, functional diagrams, structure diagrams, data diagrams, and behavior diagrams. Requirement diagrams can describe different stakeholders' need including modeling purposes, simulation technologies, constrains of simulation target and etc. Then based on requirement we can select different feature model to describe design targets, simulation targets, optimization targets, system descriptions and simulation settings. System description diagrams can connect to different function model then extend to component diagrams of structure models. Each component block maps to the specified physical system model on the forth layer. Different simulation target and optimization target block can be achieved corresponding verification model and optimization model which are constructed by behaviors model. Each behavior model connects to the corresponding set of services to run the simulation process automatically. Simulation setting models in feature diagram can describe all the information during simulation including simulation type, tool selection, communicational time step point, solver type and so on. Each data model represents sets of parameter for the corresponding physical system model.

D. *Physical model and data model layer (Technology layer)*

In this paper, each sub-system component model can be uploaded into the library as a FMU or a component of co-simulation model. A master engine should be designed to master the co-simulation procedure between different physical system models. Now the solver of Matlab\Simulink is used to control the co-simulation and S-function is designed as an interface. Each physical system model has its own connections to the component diagrams in the third layer [10].

E. *Comparion with other modelling frameworks or projects*

1) *C2WT Framework*

C2WT (Command and Control Wind Tunnel) is a model-based multi-model integration platform which is from a project of American Air Force Research Laboratory. Actually, it's a framework based on Run-Time Infrastructure (RTI) of HLA [11]. In [12], this framework can support RTI to control FMU to run the co-simulation. They have their domain specified language (DSL) to configure and control the simulation. However, the DSL can only describe the model structures and configuration which cannot satisfy the demand for information integration and MBSE approach.

2) *DESTECS Project*

DESTECS (Design Support and Tooling for Embedded Control Software) is an EU project developed for fault-tolerant embedded systems design. They have a co-simulation engine to control co-simulation process between 20-sim for continuous system model and Overture tools based on Vienna Development Method for discrete-event models. An integrated development environment has been developed to integrate such physical system models [13]. However, this project has limitation for embedded system and is short for information integration during the whole life cycle.

3) *WSAF Project*

WSAF (Whole-of-System Analytical Framework) is a project running by Australia and New Zealand. In [14], the author introduces a model-based way to share contractual contents between acquirers and suppliers and provides several requirements for such technology. In this paper, only architectural model including requirements and functions has been used for information exchange. The physical system model has not been used for information exchange which means integrated system performance cannot be analyzed in the initial phase of the whole life cycle.

4) *ModelicaML*

ModelicaML is a UML profile and a language extension for Modelica. It's a good way to connect requirements with physical system simulation by Modelica language to verify if the system can satisfy the requirements which are built by UML[15]. In general, based on several industrial investigations, many industrial firms like to choose the co-simulation way for their model-integration rather than Modelica language. This is because time and financial consumption for training programs and model-rebuild. So this method has practice restrictions which will influence on its future application.

5) CERTI

CERTI is an HLA RTI developed since 1996 by ONERA, the French Aerospace Lab. It's an open source project for HLA usage. There is no DSL support, but an open source RTI is provided in this project.

6) CRYSTAL

The ARTEMIS Joint Undertaking project CRYSTAL (CRitical sYSTem engineering AcceLeration) takes up the challenge to establish and push forward an Interoperability Specification (IOS) and a Reference Technology Platform (RTP) as a European standard for safety-critical systems[18].

This project has some investigation to show tool-integration's consistency, the methodology of co-simulation was also covered to be tested and applied within the frame.

From such projects, we can find such framework which can cover models and tool-integration is really needed for current industries. As we see, SPIT framework has a more clearly structures for MBSE technologies, data-integration, tool-integration and service-oriented web deployments.

V. User Case For Physical System Model Description

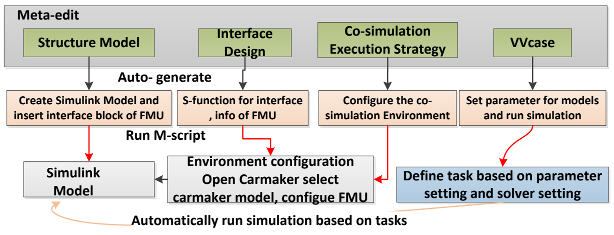

Meta-edit
Structure Model
Interface Design
Co-simulation Execution Strategy
VVcase
Auto- generate
Create Simulink Model and insert interface block of FMU
S-function for interface , info of FMU
Configure the co-simulation Environment
Set parameter for models and run simulation
Run M-script
Simulink Model
Environment configuration Open Carmaker select carmaker model, configue FMU
Define task based on parameter setting and solver setting
Automatically run simulation based on tasks

FIGURE 7 SIMULATION DESCRIPTION MODEL AND CO-SIMULATION EXECUTION

We want to use a test case to understand the clear requirements for SPIT framework. So we use MetaEdit to build the SDL and use the code-generator in MetaEdit to produce the M-script in Matlab to run the co-simulation execution automatically.

So we use the test case mentioned in Chapter 3 to build a Simulation Description Model to capture the information for co-simulation process as shown in Figure 9. Function design architecture diagram demostrated the structure of co-simulation model. Interface design diagram showed the interface for the assigned FMU. Co-simulation Strategy diagram was created to configure co-simulation environment and VVcase diagram presented the parameter setting and solver setting for each task in this scenarios. Then Meta-edit uses such diagram model to automatically produce the M-script to control the Co-simulation. Function design architecture diagram produced the M-script to create Simulink models and insert the FMU block in Simulink. Interface design diagram produced the S-function for the assigned FMU and set the FMU block in Simulink. The M-Script which produced by Co-simulation strategy diagram was used to execute co-simulation environment configuration. VVcase diagram produced the M-Script for parameter setting in Simulink Model.

A. *Testcase for SDL in MetaEdit*

We have four models in MetaEdit as Figure 9 shows. In Figure 10, the model describes the co-simulation execution strategy as operation model. The information of FMUs, subsystem models and tools which will be used in this co-simulation will be described and now we can fill such information by hand.

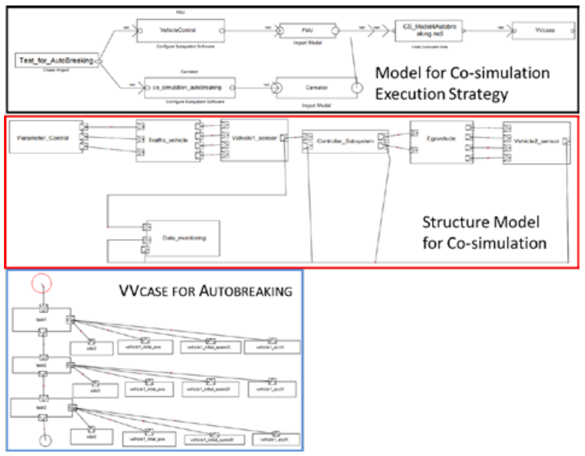


FIGURE 8 MODELS IN META-EDIT

As Figure 10 shows, the top structure model in Simulink for this autobreaking user case is built in MetaEdit. Each block describes a subsystem block in Simulink which has been put in the Simulink library.

The VVcase model in Figure 10 describes different solver settings and parameters for the same co-simulation model. In each task, different solver and parameter configurations can be finished by the parameter blocks.

In Figure 10, the interface model is used to describe to S-function for FMUs which will be used for the autobreaking test case. It includes the methods for the S-functions and all the function codes in each method. We need to fill some information for the interface by hand to produce the s-function code automatically.

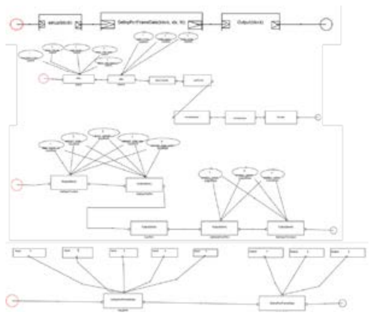


FIGURE 9 INTERFACE MODEL FOR FMUS

B. *Automatic Execution for co-simulation*

We design the Code-generator in the MetaEdit. Based on the models we built in the Meta-Edit and the information we fill by hand, the M-script in Figure 9 will be produced.

C. *Physical system model in Simulink*

In Figure 11, a Simulink model has been automatically produced by M-script and block library which we built.

Then we run the M-script produced by VVcase and co-simulation task is running automatically. During co-simulation process, Carmaker model is running as Figure 12. A front-end window has been designed for verifying the simulation result online and simulation result data has been stored in .mat files as Figure 13.

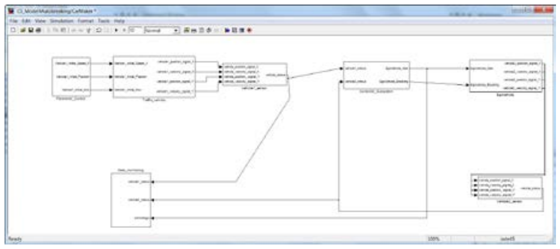
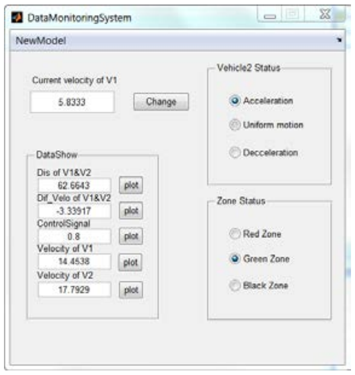


FIGURE 10 SIMULINK MODE FOR AUTOBREAKING TESTCASE

FIGURE 11 CARMAKER MODELS



DataMonitoringSystem

NewModel

Current velocity of V1: 5.8333 [Change]

DataShow
Dis of V1&V2: 62.6643 [plot]
Dif_Velo of V1&V2: -3.33917 [plot]
ControlSignal: 0.8 [plot]
Velocity of V1: 14.4538 [plot]
Velocity of V2: 17.7929 [plot]

Vehicle2 Status
[✓ Accaleration]
Uniform motion
Decceleration

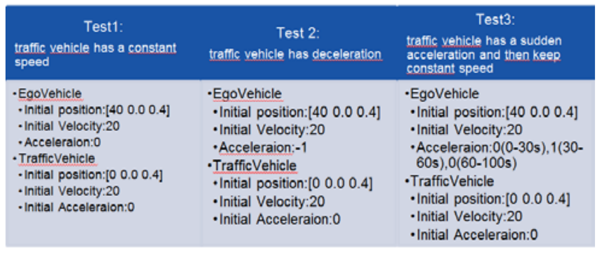
Zone Status
Red Zone
[✓ Green Zone]
Black Zone

FIGURE 12 FRONT-END FOR VERIFICATION FOR AUTOBREAKING CASE

D. *Result*

As Table 2 shows, we have three different tasks for 3 tests. Each test has different parameters for each models based on various situations. After co-simulation executions, results are shown as Figure 17.

TABLE 2 TEST CASE FOR AUTOBREAKING CASE



Test1: traffic vehicle has a constant speed	Test 2: traffic vehicle has deceleration	Test3: traffic vehicle has a sudden acceleration and then keep constant speed
•EgoVehicle •Initial position:[40 0.0 0.4] •Initial Velocity:20 •Acceleraion:0 •TrafficVehicle •Initial position:[0 0.0 0.4] •Initial Velocity:20 •Initial Acceleraion:0	•EgoVehicle •Initial position:[40 0.0 0.4] •Initial Velocity:20 •Acceleraion:-1 •TrafficVehicle •Initial position:[0 0.0 0.4] •Initial Velocity:20 •Initial Acceleraion:0	•EgoVehicle •Initial position:[40 0.0 0.4] •Initial Velocity:20 •Acceleraion:0(0-30s),1(30-60s),0(60-100s) •TrafficVehicle •Initial position:[0 0.0 0.4] •Initial Velocity:20 •Initial Acceleraion:0

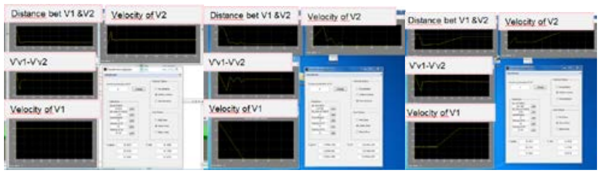


FIGURE 13 SIMULATION RESULT FOR 3 TESTS

VI. DISCUSSION

In the whole lifecycle collaboration of the SPIT framework for MBSE, several sections, such as design space exploration, system validation, searching for models, will need model management, information exchange and data management. We take an example of scenarios which OSLC can be used for the whole SPIT framework. As Figure 18 as an example, if we use the web deployments to execute the design process for SOS by co-simulation, the information exchange, co-simulation trigger, and simulation result verification can be achieved by OSLC technology.

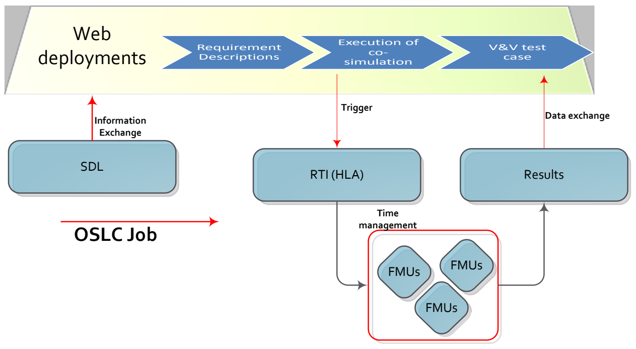


FIGURE 14 SCENARIOS FOR OSLC JOBS

VII. CONCLUSION AND FUTURE WORK

From the test case, we can see Simulation Description Modeling Language can be used to describe the basis information for co-simulation. The structure model captures the components in top system model which connected with all the subsystem models. Integrated behavior model describes the procedure of co-simulation. VVcase shows a scenario for the autobreaking test case with parameter setting and simulation setting. Interface Design Model demonstrates the approach of interface design. That means graphical model can be used to describe the co-simulation process and control the simulation running automatically. In the future, we will improve our project from four aspects. Firstly, OSLC will be adapted in this framework in order to strength the capabilities for information exchange between different models and layers. Secondly, a special master (RTI) will be designed to control different FMUs and commercial software for co-simulation [11]. Thirdly, a domain specified modeling language will be designed to capture all the information for co-simulation. The last, requirements from business process model should be dependent to ones in simulation description model.

References

[1] Martin Törgren, Dejiu Chen , Diana Malvius , and Jakob Axelsson, Model-Based Development of Automotive Embedded Systems

[2] Jinzhi. Lu "Co-simulation for heterogeneous simulation system and application for aerospace", Master. dissertation, Univ. Huazhong University of Science and Technology, Wuhan, China, 2011.

[3] Modelica® - A Unified Object-Oriented Language for Systems Modeling, https://www.modelica.org/, 2012

[4] HLA Tutorial, http://www.pitch.se/hlatutorial

[5] FMI specification, MODELISAR, https://www.fmi-standard.org

[6] OMG Systems Modeling Language (OMG SysML™), http://www.omg.org.

[7] EAST-ADL-Specification_V2.1.12,EAST-ADL Association, http://www.east-adl.info/

[8] https://www.eclipsecon.org/france2014/session/arcadia-capella-field-proven-modeling-solution-system-and-software-architecture-engineering

[9] Dong Zhang, Jin-zhi Lu , Lin Wang , and Jun Ii, Research of Model-based Aeroengine Control System Design Structure and Workflow, Asia-Pacific International Symposium on Aerospace Technology, 50(4): 511–515, 2014.

[10] Hillary Sillitto. Design Principles for Ultra-Large Scale (ULS) Systems. In INCOSE, International Symposium (INCOSE'2010, Proceedings, pages 63–82, 2010.

[11] Jinzhi Lu, Jian-wan Ding, Fanli-Zhou, Research of Tool-Coupling Based Electro-Hydraulic System Development Method, IEEE International Conference on Industrial Engineering and Information Technology, 2014.

[12] Roth K E, Barrett S K. Command & control wind tunnel integration and overview[C]//Proceedings of the 2009 SISO European Simulation Interoperability Workshop. Society for Modeling & Simulation International, 2009: 45-51.

[13] Neema H, Gohl J, Lattmann Z, et al. Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems[C]//10th International Modelica Conference. 2014: 10-12.

[14] Broenink J F, Kleijn C, Larsen P G, et al. Design support and tooling for dependable embedded control software[C]//Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems. ACM, 2010: 77-82.

[15] Do Q, Cook S, Lay M. An Investigation of MBSE Practices across the Contractual Boundary[J]. Procedia Computer Science, 2014, 28: 692-701.

[16] Schamai W, Fritzson P, Paredis C J J, et al. ModelicaML value bindings for automated model composition[C]//Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium. Society for Computer Simulation International, 2012: 31.

[17] Noulard E, Rousselot J Y, Siron P. CERTI, an Open Source RTI, why and how[C]//Spring Simulation Interoperability Workshop. 2009: 23-27.

[18] http://www.crystal-artemis.eu/

[19] OSLC specification, http://open-services.net/resources/

Session 25

Domain Specific Language 1

Friday 29th, 09:00 – Auditorium St Exupery

A Tool-Supported Approach for Concurrent Execution of Heterogeneous Models

Benoit Combemale[1], Cédric Brun[2], Joël Champeau[3], Xavier Crégut[4], Julien Deantoni[5], and Jérome Le Noir[6]

[1] Inria and Univ. Rennes 1, France
benoit.combemale@inria.fr
[2] Obeo, France
cedric.brun@obeo.fr
[3] ENSTA Bretagne, France
joel.champeau@ensta-bretagne.fr
[4] INPT ENSEEIHT, IRIT, France
xavier.cregut@enseeiht.fr
[5] University of Nice Sophia Antipolis, I3S/INRIA AOSTE, France
julien.deantoni@polytech.unice.fr
[6] Thales Research & Technology, France
jerome.lenoir@thalesgroup.com

1 Context and Approach

In the software and systems modeling community, research on domain-specific modeling languages (DSMLs) is focused on providing technologies for developing languages and tools that allow domain experts to develop system solutions efficiently. Unfortunately, the current lack of support for explicitly relating concepts expressed in different DSMLs makes it very difficult for software and system engineers to reason about information spread across models describing different system aspects [4].

As a particular challenge, we investigate in this paper relationships between, possibly heterogeneous, behavioral models to support their concurrent execution. This is achieved by following a modular executable metamodeling approach for behavioral semantics understanding, reuse, variability and composability [5]. This approach supports an explicit model of concurrency (MoCC) [6] and domain-specific actions (DSA) [10] with a well-defined protocol between them (incl., mapping, feedback and callback) reified through explicit domain-specific events (DSE) [12]. The protocol is then used to infer a relevant behavioral language interface for specifying coordination patterns to be applied on conforming executable models [17].

All the tooling of the approach is gathered in the GEMOC studio, and outlined in the next section. Currently, the approach is experienced on a systems engineering language provided by Thales, named Capella[7]. The goal and current state of the case study are exposed in this paper.

[7] Cf. https://www.polarsys.org/capella/

2 The GEMOC Studio

The GEMOC Studio is an eclipse package that contains components supporting the GEMOC methodology for building and composing executable Domain-Specific Modeling Languages (DSMLs). It includes two workbenches: the *GEMOC Language Workbench* and the *GEMOC Modeling Workbench*. The language workbench is intended to be used by language designers (aka domain experts), it allows to build and compose new executable DSMLs. The Modeling Workbench is intended to be used by domain designers, it allows to create and execute heterogeneous models conforming to executable DSMLs.

The GEMOC Studio results in various integrated tools that belong into either the language workbench or the modeling workbench. The language workbench put together the following tools seamlessly integrated to the Eclipse Modeling Framework (EMF: `https://eclipse.org/modeling/emf`):

- *Melange* (`http://melange-lang.org`), a tool-supported meta-language to modularly define executable modeling languages with execution functions and data, and to extend (EMF-based) existing modeling languages [10].
- *MoCCML*, a tool-supported meta-language dedicated to the specification of a Model of Concurrency and Communication (MoCC) and its mapping to a specific abstract syntax of a modeling language [6].
- *GEL*, a tool-supported meta-language dedicated to the specification of the protocol between the execution functions and the MoCC to support feedback of the runtime data and to support the callback of other expected execution functions [12].
- *BCOoL* (`http://timesquare.inria.fr/BCOoL`), a tool-supported meta-language dedicated to the specification of language coordination patterns, to automatically coordinates the execution of, possibly heterogeneous, models [17].
- *Sirius Animator*, an extension to the model graphical syntax designer Sirius (`http://www.eclipse.org/sirius`) to create graphical animators for executable modeling languages[8].

The different concerns of an executable modeling language as defined with the tools of the language workbench are automatically deployed into the modeling workbench that provides the following tools:

- *A Java-based execution engine* (parameterized with the specification of the execution functions), possibly coupled with TimeSquare (`http://timesquare.inria.fr`) [9] (parameterized with the MoCC), to support the concurrent execution and analysis of any conforming models.
- *A model animator* parameterized by the graphical representation defined with Sirius Animator to animate executable models.
- *A generic trace manager*, which allows system designers to visualize, save, replay, and investigate different execution traces of their models.

[8] For more details on Sirius Animator, we refer the reader to `http://siriuslab.github.io/talks/BreatheLifeInYourDesigner/slides`

– *A generic event manager*, which provides a user interface for injecting external stimuli in the form of events during the simulation (e.g., to simulate the environment).
– *An heterogeneous coordination engine* (parametrized with the specification of the coordination in BCOoL), which provides runtime support to simulate heterogeneous executable models.

The GEMOC studio is open-source and domain-independent. The studio is available at `http://gemoc.org/studio`

3 Industrial Case Study: xCapella

Arcadia (`https://www.polarsys.org/capella/arcadia.html`) is a model-based engineering method for systems, hardware and software architectural design. It has been developed by Thales between 2005 and 2010 through an iterative process involving operational architects from all the Thales business domains. Arcadia promotes a viewpoint-driven approach (as described in ISO/IEC 42010 Systems and Software Engineering - Architecture Description [1]) and emphasizes a clear distinction between need and solution. The *Capella* modeling workbench is an Eclipse application implementing the ARCADIA method providing both a DSML and a toolset which is dedicated to guidance, productivity and quality. The Capella DSML aggregates a set of 20 metamodels and about 400 meta-classes involved in the five engineering phases (*aka.* Architecture level) defined by ARCADIA. The *Capella* modeling workbench is based on Sirius in order to define the graphical concrete syntax of the Capella DSML. *Capella Studio* provides a full-integrated development environment, which aims at assisting the development of extensions for Capella modeling workbench. This studio is based on Kitalpha incubated at Thales for several years before being recently released in open source as one of the PolarSys projects. Kitalpha allows viewpoint designers to extend the Capella DSML. Despite the existence of behavioral models, the Capella modeling workbench does not provide any simulation capability. The Capella behavioral models are limited to: modes and states, functional chain data flows, and scenarios. Neither the behavioral semantics or the coordination between these languages are defined.

3.1 Objectives and overcoming initial limitations

In order to support the execution of models, a dedicated executable concurrent semantics is required. We started with two of the three behavioral languages from the Capella DSML (*i.e.*, data flows and mode automata).

In addition, to capture the interaction between the models conforming these behavioral languages, we specified the behavioral coordination patterns between them (Fig. 1).

Our study proposes to use the GEMOC studio to support system engineers so they can tame system modeling activity and improve the confidence in the

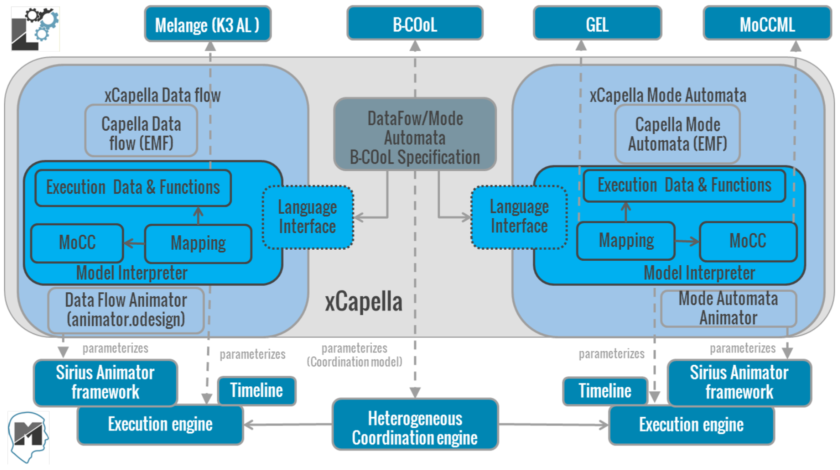

Melange (K3 AL)
B-COoL
GEL
MoCCML
xCapella Data flow
Capella Data flow (EMF)
Execution Data & Functions
MoCC
Mapping
Model Interpreter
Data Flow Animator (animator.odesign)
Language Interface
DataFow/Mode Automata B-COoL Specification
xCapella
xCapella Mode Automata
Capella Mode Automata (EMF)
Mode Automata Animator
parameterizes
parameterizes (Coordination model)
Sirius Animator framework
Timeline
Execution engine
Heterogeneous Coordination engine

Fig. 1. xCapella

specification of the system to be built. Our goal is to reduce the risks concerning inconsistent functional requirements by providing a simulation environment of the existing specification, suitable to understand/analyse the system behavior.

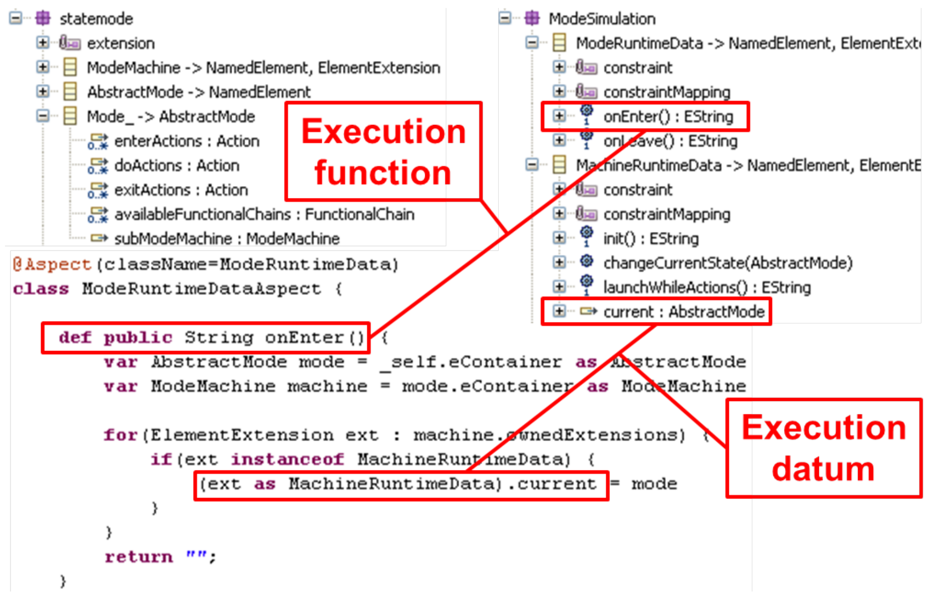
3.2 Current experimentation

This section presents our approach to design the concurrency-aware xDSMLs of Capella. This experiment relies on the Capella metamodel (which is publicly available[9]) augmented with a dedicated extension for mode automata. This mode automata extension has been done by using Kitalpha, integrated to the Capella studio.

The GEMOC language workbench

Definition of behavioral semantics: To provide a behavioral semantics, we defined the semantics in two steps: (1) an extension of the metamodel with execution function and execution data and (2) the concurrent control flow definition. In this section, we focus on the definition of the mode automata semantics. The data flows semantics is not shown in this article but is used in the coordination pattern specification defined later in this paper. The mode automata DSML (Fig. 2 at the left side) has been extended with kitAlpha in order to add classes, attributes, references specifying the Execution Data (ED) (Fig. 2 at the right side). With the *Melange* tooling support, the mode automata DSML is extended by the definition of the execution functions, which define the sequential part of the mode automata operational semantics. Melange weaves the additional operation implementations specifying the execution functions (Fig. 2 at the bottom side).

[9] `https://www.polarsys.org/projects/polarsys.capella`


statemode
extension
ModeMachine -> NamedElement, ElementExtension
AbstractMode -> NamedElement
Mode_ -> AbstractMode
enterActions : Action
doActions : Action
exitActions : Action
availableFunctionalChains : FunctionalChain
subModeMachine : ModeMachine
Execution function
ModeSimulation
ModeRuntimeData -> NamedElement, ElementExt
constraint
constraintMapping
onEnter() : EString
onLeave() : EString
MachineRuntimeData -> NamedElement, ElementE
constraint
constraintMapping
init() : EString
changeCurrentState(AbstractMode)
launchWhileActions() : EString
current : AbstractMode
Execution datum

```
@Aspect(className=ModeRuntimeData)
class ModeRuntimeDataAspect {

    def public String onEnter() {
        var AbstractMode mode = _self.eContainer as AbstractMode
        var ModeMachine machine = mode.eContainer as ModeMachine

        for(ElementExtension ext : machine.ownedExtensions) {
            if(ext instanceof MachineRuntimeData) {
                (ext as MachineRuntimeData).current = mode
            }
        }
        return "";
    }
```

Fig. 2. GEMOC-xCapella: Definition of execution functions and data

The execution functions are orchestrated by the definition of a data-independent concurrent control flow (the data-dependent aspects of the control flow are encapsulated in the execution functions). In our approach, this control flow is captured in the so called Model of Concurrency and Communication (MoCC). The MoCC is a set of DSEs, specifying at the language level how, for a specific model, the event structure defining its concurrent control flow is obtained. The event structure represents all the possible execution paths of the model (including all possible interleavings of events occurring concurrently). For the definition of this control flow we used MoCCML [7] to specify our MoCC. MoCCML is a declarative meta-language designed to express constraints between events. The constraints can be capitalized into some libraries that are agnostic of any abstract syntax. The MoCC is compiled to a Clock Constraint Specification Language (CCSL) model interpreted by the TimeSquare tool [9]. The definition of the DSEs is realized by using the Event Constraint Language (ECL [8]), an extension of OCL which allows the definition of DSE in the context of concepts from the metamodel (see listing 1.1 where DSEs *entering* and *leaving* are defined in the context of an *AbstractMode*). Finally, the behavioral semantics is obtained by using a *Communication Protocol* which maps some of the DSEs from the MoCC to the execution functions (see Listing 1.1 where the DSEs are mapped to the execution functions *onEnter()* and *onLeave()* defined in the extension of Figure 2). This means that at the model level, when an event occurs, it triggers the execution of the associated execution function on an element of the model. Currently the implemented communication protocol is quite simple but *GEL* [12] can be used to support more complex communication, for instance to specify data-dependent control.

Listing 1.1. Partial ECL specification of the mode automata

```
package statemode
 context AbstractMode
  def : entering : Event = self.ownedExtensions->select(E |
                                E.oclIsTypeOf(ModeRuntimeData))->first().
                                onEnter()
  def : leaving : Event = self.ownedExtensions->select(E |
                                E.oclIsTypeOf(ModeRuntimeData))->first().
                                onLeave()
```

Definition of the animation layer: We provide a new Sirius specification model (animator.odesign) which defines how the model representations change during the simulation. The animator is an extension of the concrete syntax definition which is part of Capella contributing a xCapella animation layer(Fig. 3) which customizes shape styles to highlight activated transition, add a decorator for the current mode and declare actions to toggle breakpoints.

The Sirius Animator framework also bring an integration with the Eclipse Debug user interface to inspect the runtime state of the execution, navigate to the corresponding diagrams and control the execution step by step.

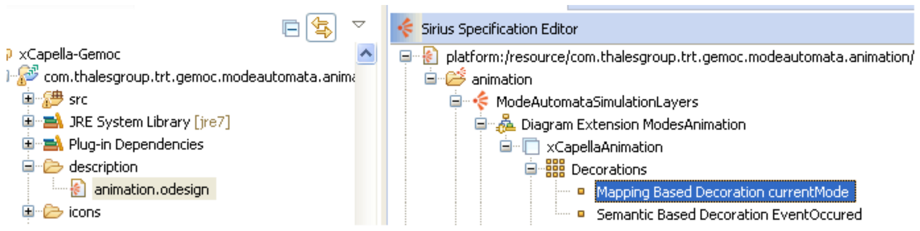

xCapella-Gemoc
com.thalesgroup.trt.gemoc.modeautomata.anima
src
JRE System Library [jre7]
Plug-in Dependencies
description
animation.odesign
icons
Sirius Specification Editor
platform:/resource/com.thalesgroup.trt.gemoc.modeautomata.animation/
animation
ModeAutomataSimulationLayers
Diagram Extension ModesAnimation
xCapellaAnimation
Decorations
Mapping Based Decoration currentMode
Semantic Based Decoration EventOccured

Fig. 3. GEMOC-xCapella animation layer

Definition of the coordination between xData-Flow and the xMode Automata: Once both the xData-Flow and the xMode Automata have been independently developed, it is of prime importance to specify the interactions of their models. This is realized by the specification of behavioural coordination pattern in BCOoL (Behavioral Coordination Operator Language).

In our case, a mode is associated to some functional chains. The coordination pattern must specify that a functional chain is activated (but not necessarily started) when the mode automata is in a specific mode. Consequently, when a mode automata is in a specific mode, the functional chains not associated to this mode are deactivated.

The BCOoL behavioral pattern contains two parts:

– a *matching*, which defines a predicate based on the DSE context to identify what are the events to coordinate in a specific model; and

– a *MoCCML constraint*, to specify how the matched events are coordinated.

In our BCOoL specification (see listing 1.2), the *ModeEnteringActivateFunctionalChain* operator coordinates the action of entering and leaving a mode with the activation of a functionalChain. Entering into a mode is identified by the entering DSE defined in the context of an AbstractMode in the *mode automa behavior language interface* (*i.e.*, in modemachine.ecl). Instances of such DSE have to be coordinated with instances of the *activate* DSE defined in the *data flow behavior language interface* (*i.e.*, CapellaDataflow.ecl). The *matching* specifies that the *entering* and *leaving* event from a mode are coordinated with the *activate* event from the functional chain only if the functional chain is referenced by the mode (in the *availableFunctionalChains* collection).

Listing 1.2. Heterogeneous coordination operator between the data flow and mode automata languages

```
BCOoLSpec  XCapellaDataFlow-xCapellaModeAutomata

ImportLib 'platform:/plugin/org.gemoc.xcapella.coordination/constraint/
     modeAutomata.moccml'

ImportInterface 'platform:/plugin/org.gemoc.xcapella.dataflow.dse/ecl/
     CapellaDataflow.ecl' as dataflow
ImportInterface 'platform:/plugin/com.thalesgroup.trt.mocc.modemachine.dse/
     ecl/modemachine.ecl' as modeautomata

Operator ModeEnteringActivateFunctionalChain (enter: statemode::entering,
     leave: statemode::leaving, activate: fa::activate)
    When:
      enter.availableFunctionalChains->exists(fc | fc = activate)
    CoordinationRule:
      enableElementWhenCurrentMode(activate, enter, leave)
end Operator;
```

The GEMOC-xCapella modeling workbench Once the xDSMLs implemented with the aforementioned tools of the language workbench, they are automatically deployed into the original *Capella* modeling workbench (integrated with the *GEMOC modeling workbench*). It results in an advanced modeling workbench integrated into the Eclipse debugger for model execution. The GEMOC-xCapella modeling workbench (Fig. 4) offers an environment for system engineers to understand/control the execution of their models with :

1. a graphical feedback of their model execution. For instance, in Figure 4, the green arrow on *initializeSystem* state represents the current state.
2. a possibility to explore several execution traces with a graphical timeline that supports step forward and step backward. The timeline and the concurrent logical step decider can be used conjointly by a designer to choose the next step in case of non determinism or concurrent events. For instance in the timeline, each vertical list of bullets represents some possible futures at this step. Also, at any time during the simulation, the designer can go back in the past to explore an alternative future.
3. a possibility to add some breakpoints to *pause* the simulation when the element carrying the breakpoint is touched (*i.e.*, when an operation is called on it).

Additionnaly, a designer can use the *execution model*, which represents the causalities and synchronizations in the model (*i.e.*, the timemodel file) to generate the state space of all possible execution traces from the concurrency point of view.

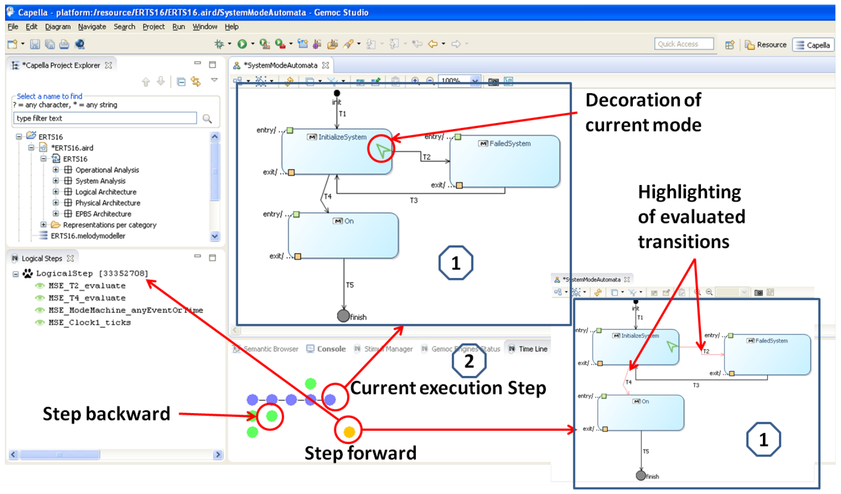


Fig. 4. GEMOC-xCapella modeling workbench

4 Related Works

In the past few years, some approaches proposed to specify the execution semantics of DSMLs by using fUML [14, 15]. While these approaches take good care to separate the execution semantics from the abstract syntax of a language, they specified the behavioral semantics as a whole by using fUML. In our approach, we use an explicit MoCC, execution functions and a protocol between them. It allows reasoning explicitly on the concurrency aspect of a language (data independently) but more important the protocol provides a natural language interface on which coordination patterns can be specified to automatically obtain coordinated simulation of heterogeneous models.

Ptolemy [11] and Modhel'x [3] also provide capabilities to simulate coordinated heterogeneous models but compared to our approach, the associated framework neither rely on a user defined abstract syntax nor on explicit coordination patterns, amenable to the easy customization of the coordination to fit the domain of use.

Finally, when some models are coordinated with our approach, it relies on both an explicit behavioral semantics and an explicit coordination. Making explicit the behavioral semantics and the coordination enables the comprehensive incorporation of semantic adaptation between the heterogeneous component. This is a major difference compared to existing approaches based on co-simulation bus (e.g. where they use the FMI/FMU standard[10]) in which the coordination is either done in the importing tool or by the manual writing of a master on the bus [2, 13]. Co-simulation bus approaches are very complementary to our approach. We believe that our approach can be used earlier in the development process, to allow, for instance synthesizing a bus master according to the explicit specification of the coordination.

5 Conclusion and perspectives

The GEMOC methods and tools have been validated through the use of an experimental (Technology Readiness Level 3) integrated advanced simulation prototype (Fig. 4). The experiment is focused on: the use of the GEMOC methodology and studio to define the behavioral semantics and coordination of *mode automata* and *data flow*; the customization of each language graphical notation for animation. The GEMOC modeling workbench provides also a well-integrated model debugging environment based on Eclipse, including advanced features for graphical model animation and execution trace management (time line). Finally, we have a proof of concept of the integration of the GEMOC execution engine and the Sirius animator framework into the Capella legacy industrial engineering workbench. The experiments result in a prototype named *xCapella*, an extension of Capella that supports the execution and animation of behavioral models. For now, even if the coordination pattern between data flows and mode automata languages has been defined, the GEMOC *heterogeneous coordination engine* [16] is not integrated to xCapella; this task is already started. Some longer terms perspectives are the definition of the behavioral semantics of the Capella scenario language and to the identification of xCapella main semantics variation points. It is also planed to provide an export of an executable model in the FMI2 standard[11].

Acknowledgement. This work is supported by the ANR INS Project GEMOC (ANR-12-INSE-0011) and The GEMOC Initiative (cf. `http://gemoc.org/ins`).

References

1. Systems and software engineering – architecture description. ISO/IEC/IEEE 42010:2011(E) pp. 1–46 (2011)
2. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., et al.: The functional mockup interface for tool independent exchange of simulation models. In: 8th International Modelica Conference, Dresden. pp. 20–22 (2011)
3. Boulanger, F., Hardebolle, C.: Simulation of Multi-Formalism Models with ModHel'X. In: Proceedings of ICST'08. pp. 318–327. IEEE Comp. Soc. (2008)

[10] http://fmi-standard.org
[11] `www.fmi-standard.org`

4. Combemale, B., Deantoni, J., Baudry, B., France, R., Jézéquel, J.M., Gray, J.: Globalizing Modeling Languages. Computer pp. 68–71 (Jun 2014), http://hal.inria.fr/hal-00994551
5. Combemale, B., Deantoni, J., Vara Larsen, M., Mallet, F., Barais, O., Baudry, B., France, R.: Reifying Concurrency for Executable Metamodeling. In: Martin Erwig, R.F.P., van Wyk, E. (eds.) 6th International Conference on Software Language Engineering (SLE 2013). Lecture Notes in Computer Science, Springer-Verlag, Indianapolis, 'Etats-Unis (2013), http://hal.inria.fr/hal-00850770
6. Deantoni, J., Issa Diallo, P., Teodorov, C., Champeau, J., Combemale, B.: Towards a Meta-Language for the Concurrency Concern in DSLs. In: Design, Automation and Test in Europe Conference and Exhibition (DATE). Grenoble, France (Mar 2015), https://hal.inria.fr/hal-01087442
7. Deantoni, J., Issa Diallo, P., Teodorov, C., Champeau, J., Combemale, B.: Towards a Meta-Language for the Concurrency Concern in DSLs. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'15). Grenoble, France (Mar 2015), https://hal.inria.fr/hal-01087442
8. Deantoni, J., Mallet, F.: ECL: the Event Constraint Language, an Extension of OCL with Events. Research Report RR-8031, INRIA (Jul 2012), https://hal.inria.fr/hal-00721169
9. DeAntoni, J., Mallet, F.: TimeSquare: Treat your Models with Logical Time. In: 50th Int. Conf. on Objects, Models, Components, Patterns. LNCS, vol. 7304, pp. 34–41. Springer (2012)
10. Degueule, T., Combemale, B., Blouin, A., Barais, O., Jézéquel, J.M.: Melange: A Meta-language for Modular and Reusable Development of DSLs. In: 8th International Conference on Software Language Engineering (SLE). Pittsburgh, United States (Oct 2015), https://hal.inria.fr/hal-01197038
11. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity – the Ptolemy approach. Proc. of the IEEE 91(1), 127–144 (2003)
12. Latombe, F., Crégut, X., Combemale, B., Deantoni, J., Pantel, M.: Weaving Concurrency in eXecutable Domain-Specific Modeling Languages. In: 8th ACM SIGPLAN International Conference on Software Language Engineering (SLE). ACM, Pittsburg, United States (2015), https://hal.inria.fr/hal-01185911
13. Marinescu, R., Kaijser, H., Mikučionis, M., Seceleanu, C., Lönn, H., David, A.: Analyzing industrial architectural models by simulation and model-checking. In: Artho, C., Ölveczky, P.C. (eds.) Formal Techniques for Safety-Critical Systems, Communications in Computer and Information Science, vol. 476, pp. 189–205. Springer International Publishing (2015), http://dx.doi.org/10.1007/978-3-319-17581-2_13
14. Mayerhofer, T., Langer, P., Wimmer, M., Kappel, G.: xMOF: Executable DSMLs based on fUML. In: 6th Int. Conf. on Software Language Engineering. LNCS, vol. 8225, pp. 56–75. Springer (2013)
15. Tatibouët, J., Cuccuru, A., Gérard, S., Terrier, F.: Formalizing Execution Semantics of UML Profiles with fUML Models. In: 17th International Conference on Model Driven Engineering Languages and Systems. LNCS, vol. 8767, pp. 133–148 (2014)
16. Vara Larsen, M., Deantoni, J., Combemale, B., Mallet, F.: A Model-Driven Based Environment for Automatic Model Coordination. In: CEUR (ed.) Models 2015 demo and posters. Models 2015 demo and posters, Ottawa, Canada (Oct 2015), https://hal.inria.fr/hal-01198744
17. Vara Larsen, M.E., Deantoni, J., Combemale, B., Mallet, F.: A Behavioral Coordination Operator Language (BCOoL). In: Lethbridge, T., Cabot, J., Egyed, A. (eds.) International Conference on Model Driven Engineering Languages and Systems (MODELS). p. 462. No. 18, ACM, Ottawa, Canada (Sep 2015), https://hal.inria.fr/hal-01182773

Towards an Ontology-Driven Framework for Simulation Model Development

Sangeeth saagar Ponnusamy[(a)], Patrice Thebault[(b)], Vincent Albert[(c)]

[(a),(b)]Airbus Operations SAS, 316 Route de Bayonne, Toulouse-31060, France
[(a),(c)]CNRS, LAAS, 7 Avenue Colonel de Roche, Toulouse-31400, France
[(a)]sangeeth-saagar.ponnusamy@airbus.com, [(b)]patrice.thebault@airbus.com, [(c)]valbert@laas.fr

An ontology driven domain model approach for improving the fidelity of the simulation by developing models through the inclusion of simulation objectives for the system Verification & Validation activities (V&V) is presented. The system V&V by simulation ontology used to build this domain model is briefly outlined in the system teleological framework of Structure, Behavior, Function, Interface and Operation. The concept of operating mode is proposed and discussed with an example. The ontology approach is demonstrated with an aircraft nacelle anti-ice system presented in the experimental frame formalism. An example of using the inference and query capabilities of the domain model approach to identify and justify abstractions consistent with the test scenarios is illustrated with a failure mode case study for this application case. The relationship with formal behavioral approach through operating modes is briefly discussed at the end where some theoretical results on the behavioral compatibility of the experimental frame components with interface simulation distances are briefly presented. The paper concludes with a discussion on the benefits of this approach from an industrial perspective along with an overview of the challenges ahead and the future work.

Keywords: *Modeling & Simulation, Ontology, Domain Model, Fidelity, MBSE, Operating Mode*

1. INTRODUCTION

In the development of complex engineering systems, Modeling and Simulation (M&S) is becoming a key capability to perform design and validation studies. However, in developing models to represent the system, often the difficulty is finding and implementing abstractions of the system being simulated, particularly with respect to the context under which it will be used. This not only leads to model validity problems identifiable only at the simulation runtime, but also results in over or under specification and sub optimal development of systems. These challenges in simulation model development necessitate a Model Based Systems Engineering (MBSE) approach which enables a common understanding by making domain assumptions explicit and separate domain knowledge from the operational knowledge. In addition, since modeling can be interpreted as a ‘reasoning’ problem i.e. inclusion of relevant information about the system being modeled, it is important to identify, relate and organize this information. However, this is often a tedious task which necessitates a domain model approach with reasoning and knowledge exploitation capabilities. Ontologies serve as a good candidate for building such a domain model approach due to their standardization in terms of OWL[1] language, scalability, and availability of tools such as Protégé[2] with SPARQL[3] query capabilities. The flexibility of ontology in expressing different domain knowledge in a succinct and standard form could significantly improve the modeling activities by explicitly incorporating the model context of usage and thereby ensuring better simulation fidelity.

1.1 STATE OF THE ART

The interest of ontologies in the M&S domain has been discussed in [Fishwick,2004] and an ontology based dictionary of generic M&S terms has been given in [Oren,2011]. Similarly, an ontology for system V&V has been proposed in [Kezadri,2010] with various formalisms and techniques for the purpose of knowledge sharing between stakeholders. However, a holistic application of ontology in simulation model development for system validation has not been explored adequately to the best of our knowledge. This study envisages such an integrated approach which consolidates knowledge capture via domain model and exploitation techniques to build a modeling abstraction library and automated assembly of model for near seamless deployment. In addition, as remarked in [Wagner,2012], [Jenkins,2012], ontologies could be used in conjunction with industry standard SysML based MBSE and this will help engineers to capitalize on the graphical syntax of SysML and reasoning capabilities of ontology.

[1] http://owl.man.ac.uk/factplusplus/
[2] http://protege.stanford.edu/
[3] www.w3.org/TR/rdf-sparql-query/

The overall ontology based approach to simulation model development is discussed in section 2 and the system V&V by simulation ontology concepts are elaborated in section 3. The classical system teleological notions of Structure, Behavior, Function, Interface (SBFI) are given in [Goel,2009] [Garo,2004]. However, these notions could be restrictive in expressing the test scenarios in the V&V context and are extended with the concept of Operation into SBFIO ontology and implemented in the Protégé tool. The Operating Modes formalism proposed in this approach is similar to mode automata [Maraninchi,1998] but is more flexible and amenable to ascribe functional or system behaviour at higher levels of abstraction. In section 3.2, extending the concept of abstraction hierarchy defined over lattice in formal verification [Cousot,1992] and semantic annotation [Lickly,2011] to V&V domains, a distance notion is ascribed to the elements of lattice since an absolute lattice inclusion relation could be too restrictive. This relative distance approach improves the application of SPARQL query capabilities of the ontology approach to the simulation model assembly [Novk,2011]. This domain model approach is briefly discussed in a process oriented perspective in section 4. An example of using the inference and query capabilities of the domain model approach to identify and justify abstractions consistent with the test scenarios is illustrated with a failure mode case study in section 5.

In addition, the concept of operating modes could serve to bridge the existing gap between the rigorous behavioral abstraction frameworks such as bisimulation [Girard,2005] and less formal system engineering approaches [Retho,2013]. In this context, a brief discussion on using ontology-aided quantitative behavioral interface refinement [Ĉerny,2010] is given in section 5 followed by a brief description of the future work and conclusion.

2. DESIGNED FIDELITY APPROACH

In the classical simulation model development process, models are usually developed independently of their context of usage and this bottom up approach often results in over or under detailed models which are inadequate to perform V&V activities. Instead of this 'measured fidelity' approach, a 'designed fidelity' approach is proposed where fidelity needs are incorporated apriori in the model development process. This necessitates collection of knowledge about the system to be modeled and scenarios under which it will be operated called System Description (SD) knowledge and Test Description (TD) knowledge respectively. In other words, SD defines the system capabilities whereas TD defines the context of usage and the expected outcomes. The system validation process normally involves interaction between system designers responsible for SD, testers i.e. simulation user responsible for TD and model developers. It is imperative for the model developer to understand and incorporate only the essential elements needed for the test and usually this set of model requirements MR is given by the model specialist. However, owing to the complexity of different domains of knowledge involved which are often implicit and incomplete, it is a tedious task to define this MR manually.

The limitations of the manual approach in its inability to handle the complexity, error prone nature, lack of archival and reuse capabilities necessitates a domain model i.e. a predefined 'template'. Such a template needs to cover the different perspectives of the knowledge description usually expressed in informal natural languages. Since a model can be interpreted as a set of concepts with some relationships between them, ontologies could be used to build such a template i.e. a domain model. This single domain model is instantiated by different actors such as the simulation user and system developer and this ontology serves as a basis for translating text based TD and SD into a standardized model to write MR. An additional advantage of ontologies is its reasoning i.e. ability to infer knowledge which is otherwise hidden or scattered. The existence of plug-in reasoners with Protégé tool such as Fact++, Hermit[4] helps to draw inferences and check consistency. Reasoners infer this relationship by reification, a concept in logic where an instance of a relation is made the subject of another relation. The inferred ontology can be queried for specific needs with SPARQL, a query language which is used to retrieve and manipulate data stored as a Resource Description Framework, RDF, a standard for the semantic web. Queries are constructed in triple pattern of subject, predicate and object with conjunctions, disjunctions and optional patterns such as to filter, sort etc. In the next section this ontology is presented followed by some applications of using of queries over them.

3. SYSTEM V&V BY SIMULATION ONTOLOGY:

In developing a domain model it is important to incorporate different viewpoints in the system teleological perspective such as SBF ontology [Garo,2004]. This can be extended with notation of interface (I) and Operating mode (O) to describe interconnected system with different modes of operation. The SBFIO ontology presented in this paper has different such generic and domain specific concepts with a modest size of about 900 triples and a part of this ontology is illustrated below in figure 1.

[4] http://hermit-reasoner.com/

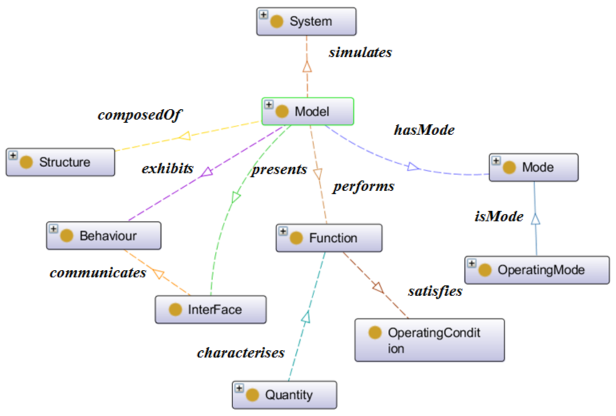


Figure 1: SBFIO Domain Model

The key concepts of the SBFIO ontology covering such a perspective are briefly given as follows:

Structure: In addition to classical architectural descriptions of how the system is built (eg: composition) [Ponnusamy,2015], spatial information is included in our domain model. Besides ensuring geometric consistency aspects, the spatial information could be related to the corresponding physical phenomenon and the interaction between the systems.
Behavior: A system behavior is the temporal evolution of the system when subjected to some scenario and behavioral abstraction will be briefly discussed in section 5.
Function: Function describes the system objectives and how they are achieved. A system's function is essentially an energy flow manipulation and ascribing domain specific laws to such flow type the phenomenon can be modeled. For example: an aircraft actuator's function is to move the control surface according to pilot's command which involves electrical to mechanical energy conversion. Based on such abstract information the associated laws can be inferred from the library developed by the domain experts.
Interface: Interface refers to how the systems interact among themselves (eg: I/O ports) or with the external user (eg:push button). Interface defines the system boundary and can have different attributes such as range, precision etc. It may also be seen as a manifestation of the observable behavior and is essential in ensuring the consistency at interconnection and composition.
Operation: Operation generally refers to the concepts of operating modes and operating condition of the EF.
Operating condition implies the conditions of environment of the SUT and is used to ascribe assumptions behind models especially at higher abstraction level. In other words it refers to the assumptions of the EF components and is used in succinctly expressing and identifying operational domains and dependencies. For example, an operating condition of a flight control system at '*takeoff*' phase implies associated assumptions for the engine performance model at this phase. In the next section, one particular concept of the domain model, namely, operational modes are explained. In [Ponnusamy,2015] a brief description of other concepts in this ontology in the context of building a model abstraction library and automating extraction of relevant abstraction has been discussed.

3.1 Operating Modes

The *Operating Modes (OM)* proposed in this paper extends the classical notion of mode-charts [Jahanian,1994], and is akin to automata. Modes are essentially partition of a system's state space and a system can have different possible modes (eg: Switch-On/off, Engine-Start/Stop). Then the OM definition is based on a simple causality relation for interconnected systems with interdependent modes (eg: Switch-On THEN Engine-Start). This definition is amenable to ascribe functional behaviour or a semantic behaviour vis à vis the system description. In contrary to the rigorous but less flexible formalisms such as mode automata [Maraninchi,1998], our definition refers to the operational manifestation of a model under a given scenario and eases the TD and SD at different levels of abstraction in a static perspective. In other words, the effects of a component's mode on other components can be observed statically and this helps in better understanding the necessary elements to be modelled whose real dynamic behaviour will be analysed later using established formalisms such as mode automata.
Let us denote a system component by C^i having modes $M_j^i \in M^i$ where i and j refers to the component id and the corresponding mode respectively. The dependency between modes are given by mode inter-connection $I_i^k: M_j^i \to \bigcup_j M_j^k$ i.e. a mode of a component, C^i may affect one or more modes of other component, C^k such that $I_i^k \subseteq \{M^i \cup$

$M^k\}$. The OM then becomes a tuple, $OM^{ik} =< C^i, I_i^k, C^k >$ and the connected modes of C^i are called guards i.e. causative and that of C^k are called states i.e. resultant. Transitions between modes occur whenever the guard mode changes. For example, consider a system with four components, $C^{i=1..4}$ each having different modes. The dependencies in between them are shown as dotted lines below in figure 2, for example, the mode M_1^1 affects M_1^2 which in turn affects M_1^4 i.e. components C^4 is dependent on C^1.

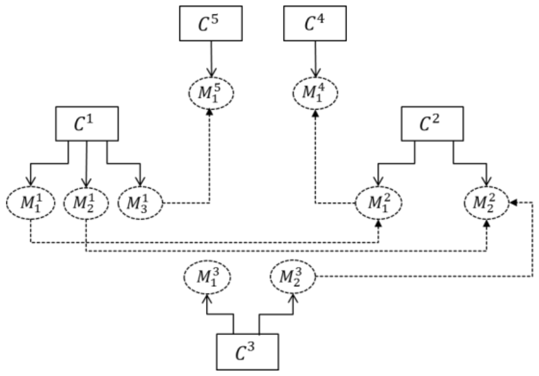


Figure 2: Mode Dependency Example

Such dependencies could be illustrated using OM in the following figure 3, which could be then reasoned and queried to find implicit information such as modes (un)affected by a particular mode or its attributes (e.g: type of system, associated designer etc). In practice the system designer need only gives the component and its dependent modes and the link between different such pairs are extracted automatically. This is useful since the designer usually knows the causality relation only few components upstream and downstream and it is thus important to relate between all such information to have holistic view before modelling the system. In other words, this helps in capturing each component's operational environment assumptions in terms of modes. In the figure below, the causality relation in mode is denoted by solid arrow line and the transition between modes by dotted arrow lines. In addition, transition can be constrained, for example, once mode M_3^1 is activated it cannot be switched to other modes of the component and hence the end state will always be M_1^5. Thus the transitions can be primary i.e. affects other OM or secondary i.e. does not affect other OM e.g.: OM$_5$.

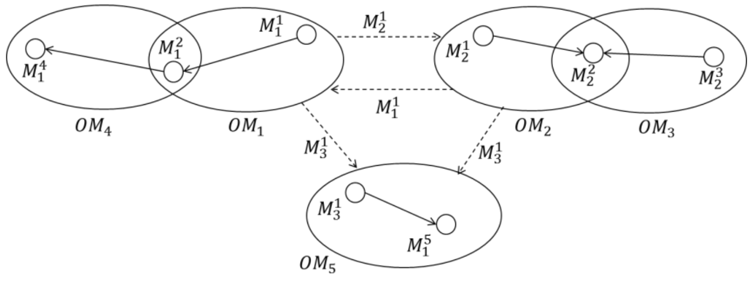


Figure 3: Operational Modes

From such illustration queries can be made on the instantiated domain model for applications such as identification of the transitions between modes and the necessary dependencies to be modelled. For example, reachability notions such as the mode M_1^4 can be reached from M_2^1 by changing the mode to M_1^1 can be queried. Similarly there are two ways of reaching M_2^2 and associated (or the shortest) path can be queried.

This description will also be useful in high level functional failure mode and effect analysis. A failure could be interpreted as the inability of the system mode to transit in response to its associated causality conditional i.e. guards change. Consider a TD stating simulate C^2 failure and this requirement necessitates inclusion of components associated to C^2 such that any mode change in upstream component i.e. guards does not have effect on C^2 since it is already failed and effect of downstream components i.e. states with respect to it. From SDD it is known that C^2 can fail at M_1^2 or M_2^2 and in case of failure at M_1^2 it can be easily infered that M_2^1 will not have any effect and it must be included to check the effect. In addition, OM$_4$ can be abstracted for simulation of OM$_5$ since it does not have any transition associated. Similarly recovery procedures such as in case of M_1^2 failure to respond to transition M_2^1, M_2^2 can be reached through OM_5 if there exists a transition i.e. guard change M_3^1 to M_2^1 can be seen. It may be reminded that all such inferences are static i.e. from instantiated domain model through queries and this helps in inclusion of necessary abstractions to be implemented for a given test requirement before dynamically simulating.

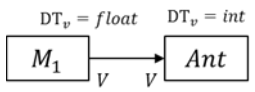
In the next section, notions of hierarchy between concepts of the domain model to build inheritance relations are discussed. These inheritance relations are then exploited to identify necessary abstractions.

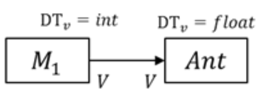
3.2 ABSTRACTION HIERARCHY

In general, a system can be modeled at different levels of abstraction which could be related to each other by a binary relation ($\preccurlyeq$). This hierarchical notion of simulation preorder represented as a lattice has been widely studied in the field of formal verification [Cousot,1992]. An application of such approach to consistency checking of semantic annotation of models has been explored in [Lickly,2011]. Our study extends such property annotation to V&V domains and ascribes a distance notion to the elements of lattice. In other words, elements of lattice which are closer to the desired element than the others have relatively higher fidelity. For example, the lattice for the variable data type concept with elements *boolean*, *int* and *float* are ordered as *boolean* $\preccurlyeq$ *int* $\preccurlyeq$ *float* where $\preccurlyeq$ refers to 'is also' relation i.e. *int* is also a *boolean* but the reverse is not true. Assuming the TD demands a variable type *float* whereas SD offers only *boolean* or *int*, intuitively it can be seen that the abstract data type *int* is closer to concrete data type *float* than the other abstract data type *boolean*. Similarly inclusion relations could well be extended to other relevant annotations such as in domain specific laws (e.g. a geopotential model with spherical harmonics is also a flat earth model), model versions etc. These inclusion relations are useful to engineers who do not necessarily share the same domain expertise. Such annotations and inclusion relations are useful in mitigating redundant modeling effort, especially in modeling system derivatives where a combination of legacy and new models will be used in tandem for the V&V activities. A similar approach could be used to document assumptions behind models in a hierarchical manner which in turn could be exploited to find the simplest consistent model meeting the simulation requirements [Ponnusamy, 2015].

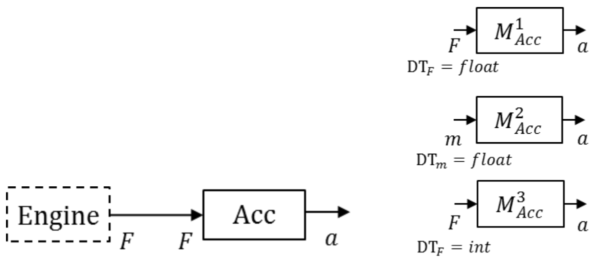
3.2.1 Automated Model Assembly

In addition to standardization of knowledge and exchange, query capabilities are exploited to select the model with consistent interfaces based on parameter matching using this distance notion. In addition, it is possible to assign weights to each attribute and the model whose interface having the closest consistency is chosen. In a component based design framework, the assembly of components is an important but often ignored aspect and many integration problems arise due to interface compatibility. In assembling i.e. connecting two models, compatible models are selected from a library of models by matching their input and output parameters of their interfaces. In [Novàk, 2011] this task is discussed via queries of ontology but the matching is exact i.e. two models are compatible only if the output of first model is same as the input of second model. This could be true for matching parameters, units etc. but for conditions such as matching data types etc. it could be stringent. Consider an example where a battery model (M_1) modeling voltage is connected via an electrical circuit to an antenna model (Ant). The battery output datatype could be '*int'* whereas the antenna model input datatype is '*float'*. A boolean type checking gives an error despite a *float* is also an *int* datatype. In our ontology, when such an instance occurs, the connection is deemed compatible as shown in figure 4(b), since in the datatype lattice described in section 4.2, *Float* $\preccurlyeq$ *Int*. This is evaluated by simply measuring the length of its relative position in the lattice chain (e.g.: *int* is located lower than *double* hence it has higher length and only elements with lower length are chosen for input type compatibility). Let us consider an example where the engine model is connected to the accelerometer (*Acc*) model to measure the acceleration, a, induced by the thrust, F. The acceleration can be calculated either as function of force or mass or both and from the set of candidate models shown in Fig.4(d) it is evident that second model cannot be used here. From the two available models the first one is chosen for its higher precision if the output datatype is the same (or better). The associated pseudo-queries for this example are given in the appendix. Similar queries can be written to match or extract other system attributes.





(a) Incompatible Assembly (b) Compatible Assembly



(c) System Assembly (d) Model Candidates

Figure 4: Model Assembly

4. PROCESS OVERVIEW

In [Ponnusamy,2015], the utilisation of such a domain model to build a model abstraction library, and automate the model selection from the library using an algorithm in SysML activity diagram is presented. The overall process of the domain model development, building and exploitation of the model abstraction library is briefly discussed in this section with an illustration of the process [Thebault,2015]. In developing a domain model, an important aspect is to describe its integration or improvement of the existing M&S process in an end user operational context. It can be seen from figure 5 that the proposed approach replaces text with domain model concepts, reasoning over implicit information to make them explicit and evaluate their consistency with respect to each other. This is followed by model selection process and the selected model is instantiated in a classical simulation tool such as Modelica etc. The phases of modeling and simulation along with the respective stakeholders can be seen from the figure below. The meta-model and selection algorithm [Ponnusamy,2015] are denoted in dotted ellipse.

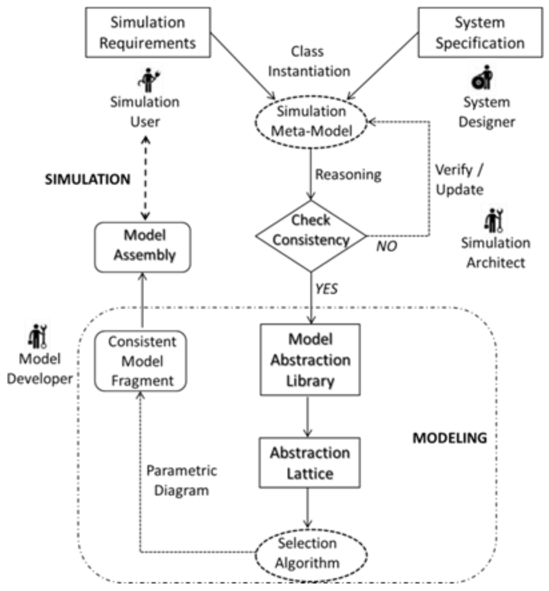


Figure 5: Operational View of M&S domain model

The meta-model instantiation by simulation users and system designers is reasoned by the simulation architect to find implicit data and evaluate its consistency to write the model requirements. In addition, the model developer documents his existing models in a library using the same meta-model, and based on the requirement from architect, a consistent abstraction is selected from this hierarchy of abstractions using the SysML algorithm and then composed with the other models. This assembled model is then deployed on a simulation platform and executed by the simulation user according to the defined V&V plan. An industrial perspective of such approach in the context of simulation fidelity is presented in [Thebault,2015]. This process can be integrated easily in the standard M&S process in industry and it can be seen that this is a non-intrusive method for the engineers since building and exploiting abstraction library is intended to be automated with minimal effort. However, as with any domain model approach in industry, initial effort will be high for tool development, workforce training, process management and deployment. But as several studies demonstrate MBSE is an important enabler in system development especially due to rapid and complex evolution of corpus of engineering knowledge in an organization and the need to capture systematically this engineering knowledge for standardization and exploitation.

5. APPLICATION CASE: AIRCRAFT NACELLE ANTI-ICE SYSTEM:

A generic description of the aircraft Nacelle Anti-Ice System (NAIS) is presented, followed by instantiation of the domain model built from the ontology defined in section 3. The NAIS is used to prevent ice accretion at the engine nacelle inlet by using hot gases from the engine exhaust. The system is comprised of controllers, valves, solenoids, ducts etc. and is connected to other aircraft systems. In order to perform tests on a component(s) (eg: controller of NAIS), the problem of selecting elements of NAIS and the associated systems (eg: Flight Management System), environment (eg: engine) with respect to this component(s) and the test scenario is outlined in the EF formalism. The following figure illustrates the environment representing the context under which the controller will be tested in the EF formalism. The general system interaction is shown by solid lines and the scenario specific observability of phenomenon (eg: pressure data from sensor) is denoted in dotted lines. Thus the EF helps in a lucid visualization of what is being tested and what is needed for the test in addition to how it is tested (controllability) and what is expected of the test (observability).

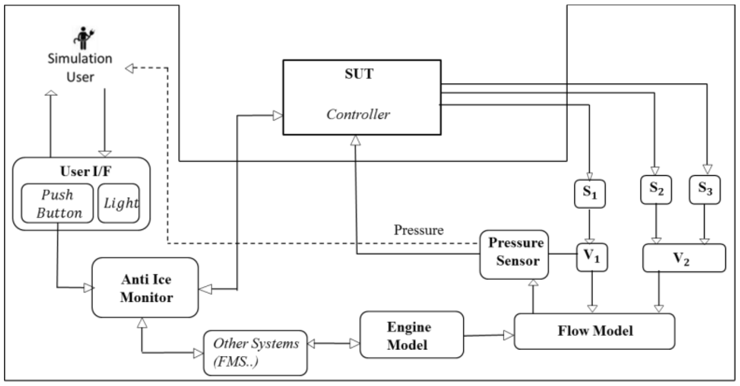


Figure 6: Experimental Frame of NAIS Controller

An application of the OM concept to the failure mode simulation of NAIS valve is illustrated in the following section similar to the example in section 3.1.

5.1 NAIS Failure Simulation

Let us consider a test scenario where TD requires the simulation of valve V_2 failure at closed mode. The test request typically says at which conditions the failure is triggered, where and what are the expected outcomes. On the other hand, SD of NAIS describes all the possible behavior of system, in this case, dependency of valve V_2 modes with the solenoid $S_{2,3}$ modes (e.g. : valve is open when solenoid is energized & closed when solenoid de-energized). It then becomes imperative to identify the components and its associated modes causally affected by this failure condition. Inferring the instantiated OM concepts and querying over this knowledge, desired information such as dependent component or the components that can be abstracted can be obtained with ease. It alleviates the burden of the tedious and often error prone task of keeping track of disparately located but hidden information which is related to each other. Following the notation given in section 3, the SD then becomes

$C^1 = \{V_2\}$	M_1^1 = open	M_2^1 = close	M_3^1 = regulating
$C^2 = \{S_2\}$	M_1^2 = de-energised,	M_2^2 = energised	
$C^3 = \{S_3\}$	M_1^3 = de-energised,	M_2^3 = energised	

The OM is built from the mode data and is illustrated below, for the sake of clarity each OM is shown separately.

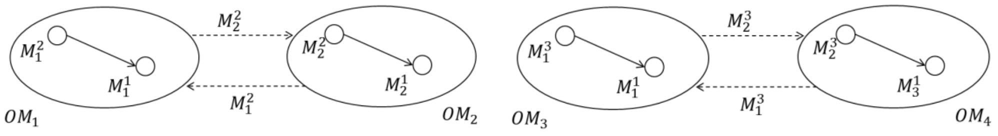


Figure 7: Operating Modes of Valve and Solenoid

Consider a test on the controller to validate its failure monitoring and reconfiguration of valves. It can be seen that, in order to simulate the valve failure when closed, it is imperative to simulate the solenoid S_3 in de-energized mode to see it does not have any effect. However this information is not explicitly given in TD as it describes expectations on the system at higher levels of abstraction whereas SD describes all possible behaviors of the system. Thus it becomes important to identify only the necessary functions and associated systems to be modeled to avoid over or under detailing of models.

In addition such an approach will help visualize and identify possible emergent behavior which may not have been modeled otherwise. For example, from the valve which is failed at the closed position, the regulating mode can be reached in two steps by having S_3 de-energised and S_2 energised. Similar extensions are possible and such information is usually not given explicitly either in SD or in TD, and this formalism helps the model specialist in writing a MR with autonomy. This particular example, though done manually, is found to increase the efficiency during test since provisions for failure triggering are explicitly identified and provided along with necessary functionalities to model the failure propagation.

6. MAPPING TO BEHAVIOURAL FRAMEWORK

The problem of quantitative transition systems and their abstraction has been widely studied by [Alfaro,2004], [Thrane,2009]. However, there exists a gap between the rigorous behavioral abstraction frameworks such as (bi)simulation relations and less formal system engineering approaches [Retho,2013]. The complexity of current engineering systems requires integration of different layers of abstraction and consistency between them. The concept of operating modes could serve as a connection between high level functional description through the domain model approach and low level behavioral description through quantitative transition system. Since OM are high level behavioral descriptions, this would lead to better identification and modeling of transitions to capture the low level behavior, especially during incremental model synthesis. Similarly, the notion of lattice distance leads naturally to behavioral distance quantification based on approximate bisimulation [Girard,2005] and simulation distances [Ĉerny,2010]. In addition, in the context of assume-guarantee contract based design [Benviste A,2012], behavioral refinement of models by extending the concept of interface with interface simulation distances [Ĉerny,2012] is also are being studied. However these studies are still in their infancy and future work includes developing the theory to bridge this behavioral approach with the semi-formal domain model approach to build a unified framework addressing the simulation needs capture from high level to low level model behavioral requirements definition.

7. CONCLUSION & OUTLOOK

This paper gives a preliminary version of domain model and the SBFIO ontology explained in this paper is being improved with other domain specific concepts in the industry and validated with stakeholders before its integration in the engineering process. The study is currently at the identification and experimentation of solutions phase and preliminary indications are encouraging. This study is also being used to assess and provide feedback to ongoing feasibility studies on using the Cappella tool based on the Arcadia framework [ARCADIA] for aircraft system architecture definition and simulation. Future work includes development of a user-friendly graphical interface for domain model instantiation, queries, and formalization of a centralized ontology management process which are all imperative for the utilization across the enterprise.

The ontology driven domain model approach helps to ensure traceability between different abstraction layers and ensures viewpoint consistency and thus enables seamless integration of models and deployment. It helps the test team to optimize the test scenario through inclusion principles and the modularization of ontologies helps in test independence to reduce redundant test combinations. It alleviates the general difficulty of the lack of synchronization and standardization between system development and testing by incrementally and iteratively improving the systems design and testing knowledge along the program schedule. This not only helps in modeling knowledge archiving and reuse for streamlined development of system variants but also for better coordination and decision making in program development.

However, it is often the case that not all abstractions are or can be documented through such a domain model as it is a time consuming and arduous task especially when multiple stakeholders are involved. This is also compounded by the fact that parsing of documents written in natural language into the domain model concepts described in section 3.2 is a complicated task in itself. Though there are some initial studies such as [Ileiva, 2005], this problem needs to be studied with cognitive techniques such as data analytics and deep mining based on iterative learning techniques for better usage of this domain model.

8. REFERENCES

Albert, V. 2009. *Simulation validity assessment in the context of embedded system design*. PhD Thesis. University of Toulouse, CNRS, LAAS, Toulouse, Unpublished.

ARCAIA, Systems Modeling with the ARCADIA method and the Capella tool, Workshop, EclipseCon 2015, Toulouse, 2015.

Benveniste A et al, "Contracts for Systems Design", Research report No:8147, INRIA, France, 2012.

Cousot, P. 1992. "Abstract Interpretation Frameworks". *Journal of Logic and Computation*, Volume 2, 511-547.

Frantz, F K. 1994. "A taxonomy of model abstraction techniques", *Proceedings of the 27th conference on winter simulation,*Arlington, Virginia, United States, 1413-1420.

Gero, J.S., Kannengiesser, U. 2004. "The situated function-behaviour-structure framework", *Design Studies*, 25(4), 373-91.

Girard A., Pappas G.J., 2007. "Approximation Metrics for Discrete and Continuous Systems". *IEEE Transactions on Automatic Control*, Volume 52, Issue 5, 782-798.

Greves, H. 2009. "Integrating SysML & OWL", *Proceedings of OWL:Experiences and Directions,* 2009.

Ilieva, M.G, Ormandjieva, O., 2005. "Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation", *Natural Language Processing and Information Systems*, Lecture Notes in Computer Science Volume 3513, 392-397.

Iwasaki, Y., Levy, A. 1994. "Automated Model Selection for Simulation". *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 108-116.

Jenkins, S., Rouqette, N. 2012. "Semantically-rigorous systems engineering using SysML and OWL", *International Workshop on System & Concurrent Engineering for Space Applications*, (Lisbon, Portugal, October 17-19, 2012).

Levy, A., Iwasaki, Y., Fikes, R. 1997. "Automated model selection for simulation based on relevance reasoning", *Artificial Intelligence*, (Nov 1997), Vol 96, Issue 2, 351–394.

Lickly, B., Shelton, C., Latronico, E., Lee, E. 2011. "A Practical Ontology Framework for Static Model Analysis", *Proceedings of the Ninth ACM international conference on Embedded software*, NY, USA, 23-32.

Man-Kit-Leung J., Mandl T., Lee E., Latronico E., Shelton C., Tripakis S., Lickly B., 2009. "Scalable semantic annotation using lattice based ontologies". *Lecture Notes in Computer Science*, Vol 5795, pages 393-407.

Natalya F. Noy., Deborah L. McGuinness. 2001. "Ontology Development 101: A Guide to Creating Your First Ontology". Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, (Mar 2001).

Novàk, P., Šindelář, R. 2011. "Applications of ontologies for assembling simulation models of industrial systems", *Proceedings of the 2011th Confederated international conference on the move to meaningful internet systems*, Vol 7046, 148-157.

Ponnusamy, S. S., Albert, V., Thebault, P. 2015. "Consistent behavioral abstractions of experimental frame", *AIAA Modeling & Simulation Technologies Conference*, July, 2015, USA, Accepted.

Ponnusamy , S. S., Albert, V., Thebault, P. 2015. "A Meta-Model for Consistent & Automatic Model Selection", *30th European Simulation and Modelling Conference*, Oct, 2015, UK, Accepted.

Thebault, P., Ponnusamy, S. S., Albert, V. 2015. "A Multimodal Approach to Simulaton Fidelity", *The 12th International Multidisciplinary Modeling & Simulation Multiconference*, Sep, 2015, Italy, Accepted.

Wagner, D.A., Bennett, M.B., Karban, R., Rouquette, N., Jenkins, S., Ingham, M. 2012. "An ontology for State Analysis: Formalizing the mapping to SysML", *IEEE Aerospace Conference,* Montana, USA, 1-16.

Zeigler B.P., Praehofer H., Tag G.K., 2000. *Theory of modeling and simulation*, San Diego, California, USA: Academic Press.

APPENDIX

```
PREFIX mm:<http://instantiated model name .owl#>
PREFIX nn:<http:// instantiated model name _infered.owl#>

#sample code to compare three simulation models input interface with system model input interface. Two of the models have
#same parameter (e.g.Force, F) but different datatypes (e.g: double, int) – first match the models having same parameters then list
#lattice length
#the query needs to be customized to suit the respective class, object and data properties respectively

SELECT  DISTINCT ?iclist ?source ?dest ?system_var_out_name ?system_var_in_name ?sim_var_in_name ?sim_block
(COUNT(DISTINCT ?sim_var_class) AS ?sim_var_class_no)
WHERE
{

#List all system Interconnection
 ?iclist rdf:type mm:Block_InterConnection;
       mm:connectsFrom ?source;
       mm:connectsTo ?dest.

#check Source Port and Destination Port have same variable names eg:F for force
?source_port a mm:SourcePort;                  owl:sameAs ?system_port_out.?system_port_out mm:isAssociatedTo
?system_param_out. ?system_param_out mm:representedBy ?system_var_out. ?system_var_out mm:hasVariableName
?system_var_name. ?system_var_name mm:hasVariableNameString ?system_var_out_name.
?dest_port a mm:DestinationPort;                  owl:sameAs ?system_port_in.
?system_port_in mm:isAssociatedTo ?system_param_in.?system_param_in mm:representedBy ?system_var_in. ?system_var_in
mm:hasVariableName ?system_var_name1. ?system_var_name1 mm:hasVariableNameString ?system_var_in_name.

#check variable datatypes

FILTER(CONTAINS(?system_var_out_name, ?system_var_in_name))
?sim_block mm:Simulates ?source;          mm:hasBlockParam ?q.
 ?q a mm:InputParameter. ?q mm:representedBy ?b. ?b mm:hasVariableName ?d.  ?b mm:hasVariableDataType ?jj. ?ii
rdfs:subClassOf* mm:VariableDataType. ?jj a ?sim_var_class. ?d mm:hasVariableNameString ?sim_var_in_name.

FILTER(CONTAINS(?sim_var_in_name, ?system_var_in_name) )
?ii rdfs:subClassOf* mm:VariableDataType.
?jj a ?sim_var_class.
}

GROUP BY ?sim_block ?iclist ?source ?dest ?system_var_out_name ?system_var_in_name ?sim_var_in_name
```

Tool Support for a Method and a Language Integrating Model Refinements and Project Management

Salma Bergaoui[1,3], Ivan Llopard[1,3], Nicolas Hili[2,3], Christian Fabre[1,3], and Fayçal Benaziz[1,3]

[1]CEA, LETI, MINATEC Campus, F-38054 Grenoble, France.
[2]LIG, F-38000, Grenoble, France.
[3]Univ. Grenoble Alpes, F-38000 Grenoble, France.

{salma.bergaoui, ivan.llopard, faycal.benaziz, christian.fabre1}@cea.fr, nicolas.hili@imag.fr

December 1, 2015

Keywords: *Embedded Systems, Development Processes, Methods & Tools, Project Management, Model-Based System Engineering, Parallelism, Action Language.*

Abstract

Complexity of Embedded System (ES) development is increasing due of several cumulative sources. Some of them are directly related to constraints on the ESs themselves, like computing power, resource constraints, and multi- or many-core programming, while other are related to the industrial context, like teamwork and parallelisation of concurrent development. In this paper we present CanHOE2, a Model-Driven Engineering (MDE) tool that addresses two issues of ES development: expression of parallelism by means of objects and Hierachical State Machines (HSMs), and teamwork synchronisation.

1 Introduction

Embedded System (ES) development teams have to cope with usual constraints of industrial organizations: (1) End-to-End Engineering: the full development cycle goes from requirement formalization to the final integration and assessment of the application on its platform. (2) Incremental & collaborative development: To organize efficiently the work of large teams, it is critical to regularly distribute and integrate work, and to measure progress towards the objectives. (3) Moreover, at any level, models should be executable and instrumented: executable to check them against requirements, and instrumented to get continuous qualitative and/or quantitative feedback to drive engineering decisions.

Another important factor of ES development efficiency is the set of modeling and programming languages used in a project. Ideally, we would rely on a single high-level modeling language that: (a) Can model hardware as well as software; (b) Is not tied to any hardware architecture like Field-Programmable Gate Array (FPGA), Digital Signal Processor (DSP) or General Purpose Processor (GPP); (c) Is parallel-friendly; (d) Provides a clear path to generate efficient code.

Lastly, the wide variety of modern platform patterns like manycore platforms with distributed memory based on Globally Asynchronous, Locally Synchronous (GALS) and Network-on-Chip (NoC) are increasingly heterogeneous and thus much more difficult to program than classical shared memory architectures. For these new architectures, parallelism has to be exposed; their distributed architecture requires strong partitioning of the code and calls for message-passing style of programming. At the same time, modern embedded applications cannot be divided between data/computational parts and control parts anymore. Instead, they are made of a number of layers that include both parallel computations on large data sets and data-dependent control as well.

In this context, we developped a Model-Driven Engineering (MDE) approach named “Highly Heterogeneous, Object-Oriented, Efficient Engineering” or $\langle\text{HOE}\rangle^2$ for short. The approach is made of:

- A *method* that provides modeling concepts necessary to describe heterogeneous embedded systems. The $\langle\text{HOE}\rangle^2$ method provides a set of related project management entities and metrics to organize, track and report on the development efforts [1, 2, 3].

- An *action language* that seamlessly combines association-based data parallelism and operations on compound data [4]. The $\langle\text{HOE}\rangle^2$ language preserves the expressiveness of Statecharts [5], and captures a layout – and implementation – neutral description of data organization, extends message passing with an intuitive semantics for this additional parallelism and provides strong foundation for array-based optimization techniques.

- A *canonical tool* named CanHOE2 that combines textual and visual programming while enforcing the principles of $\langle$HOE$\rangle^2$ for efficient management of projects. It is canonical in the sense that it aims at illustrating the main points of the method.

This paper is structured as follow. We review some related works in Section 2. Section 3 introduces the context of our approach. In Section 4, we present the CanHOE2 tool, its main features, design choices and implementation. Section 5 presents its application on a specific case study, and we conclude in Section 6 with perspectives and future work.

2 Related Work

Over the past years, many methods and tools were proposed in order to model embedded systems [6, 7, 8].

ACCORD/UML is a model-based method proposed to model real-time application models in the automative area [6, 7]. It is organized around waterfall lifecyles and implies several stakeholders such as car and parts manufacturers.

BIP (for *Behavior, Interaction and Priority*) is a framework for composing hierarchical systems [8]. It permits to build composite systems by hierarchically assembling atomic components that are described in terms of behavior and interactions. BIP defines an activity-based process in which a few set of activities is defined, for designing application models, integrating platform's constraints, generating code and for verifying the system.

MopCom is a model-based development method for designing ESs [7]. It proposes a top-down process divided up into two flows for enhancing the development parallelism, and several iterations to refine an application model onto several abstraction layers of a platform.

In all these methods, processes are defined alongside generic or specific tools. Tools are used in order to support both the language used for modeling ESs, and the process with its different features (parallelism, multi-roles, etc.). However, there are several drawbacks. Tools are often generic and do not cover all the activities of an ES process. Coupling several tools can permit to cover the whole process, yet is inefficient for modeling ESs. Using non-dedicated tools hinders the development as they do not provide a global and coherent view of the developed system throughout the whole process. Hence, they allow designers to digress from the canonical process. In addition to that, they do not integrate any project management features a project manager could benefit from in order to have a global understanding of a current development and to monitor it. There is now a need of developing Integrated Development Environments (IDEs) that address all these issues in one go.

On the tool side, "IBM Rhapsody", from IBM/Rational, provides an IDE to develop UML or SysML models that is widely deployed in the industry [9]. The dynamic behavior of an object is captured in a flavor of Finite-State Machine (FSM) where the transitions' actions are called by external libraries developed in a general-purpose language. The FSM can be translated into C++ code, compiled and executed. During development, this code calls back into the IDE for debug and animation of the execution, *e.g.* with sequence diagrams. Several Board-Support Packages (BSPs) are also available such that the generated code can be compiled, uploaded and executed on various embedded platforms. Other tools from IBM, such as "Rules Composer" permit to define specific code generators from Rhapsody models [10]. The mixed semantics of these tools, with FSM and C++ libraries, do not allow for symbolic reasoning at the model level, be it for optimization, correctness analysis or model refinement. Although they do support MDE, they main target is executable code production.

"Papyrus MDT" [11, 12] is a component of the Eclipse Model Development Tools (MDT) project. It is a complete solution for UML modeling and is fully compliant with the latest versions of the UML standard. It supports the definition of profiles, and their applications to tailor UML models to a particular domain. Papyrus provides some tools to customize the whole modeling environment of Eclipse (i.e. editors, palettes, the model explorer and the property panel) according to the definition of the profile. Thus, designers can design models in a dedicated environment using specific profiles. However, the customization is only performed regarding the dedicated language and disregarding any process that could be combined to the dedicated language. Hence, Papyrus could be tailored to a specific language, but not to a specific method. From the model execution side, Papyrus provides code generation engines for C++ and Java. However, it focuses on structural elements and only provides a limited support for code generation from behavior models through "Qompass Designer" [13]. That means, as the authors state [14], Qompass Designer only supports simple FSM models and it is currently not possible to produce code from hierarchical FSM models.

3 Context

CanHOE2 supports and implements the $\langle$HOE$\rangle^2$ method.

3.1 The $\langle$HOE$\rangle^2$ Method

$\langle$HOE$\rangle^2$ stands for Highly Heterogeneous Object Oriented Efficient Engineering. The $\langle$HOE$\rangle^2$ method is organized around four models and three refinements, that defines a set of successive activities with clear inputs and outputs – See Fig. 1 [1, 2, 3]. These models are built as follows: (A) The informal requirements are formalized in the *Requirement Model* by means of *System*, *Actors*, *Use Cases* and *Scenarios*; (B) This requiement model is then refined into an executable, platform independent, *Analysis Model*. This is done by means of *Hierarchical Opening* of objects: replacing an object viewed as black box with several object that represent a more detailed version of the orginal object while collectively exhibiting the same apparent behavior. This transformation can be applied recursively to new objects. (C) The *Platform* is partially introduced by declaring its *Worlds* – See (1) in Fig. 1. A world is an abstraction of an execution domain that can *Hosts* objects – like a set of processors and their shared memory, a FPGA, or a dedicated hardware Intellectual Property (IP). Once the platforms's worlds are known, the *Design Model* is built. For this, each object from the analysis model is split in smaller objects that are *Distributed* over the platform's worlds. (D) Further platform details are introduced into the application: for each world, its *Containers* are provided. Each container embodies a set of coding rules, with a trade-off, to implement an object semantics on the target world. The *Implementation Model* is built by *Injecting* each object of the design model into a container.

In terms of associated process management, we define the following concepts:

- *Task*, *Phase* and *Task Sheets*: A task is an atomic modeling activity: one of the three refinements applied to a single objet. A phase is the set of tasks that captures all the refinements that build the phase's model. A task sheet is an instance of a task.
- *Participant*: A participant may be either a Project Manager (PM) or a Developer. A PM creates and drives the project by (a) defining and assigning tasks to Developers, and (b) by integrating, or rejecting, the models they produce. A Developer can modify models only according to a task sheets provided by the PM.
- *Project*: A project contains its task sheets and the four phases of the process. A project is led by a single PM and built by several Developers.
- *Iteration*: An iteration is a collection of tasks. It is the smallest entity of project management.

Table 1 illustrates an example of a consistent planning using the iterations as a diagram similar to a

Table 1: Consistent Planning of developments

		RA	SA	SD	SI
UC_1	SN_1^1	T_1^1	T_1^2	T_3^6	T_4^8
UC_2	SN_2^1				
UC_3	SN_3^2	T_2^3	T_2^4	T_3^7	
	SN_3^3				
UC_4	SN_4^1				
UC_5	SN_5^1		T_2^5		
	SE_5^2				
	SE_5^3				
UC_6	SN_6^1				

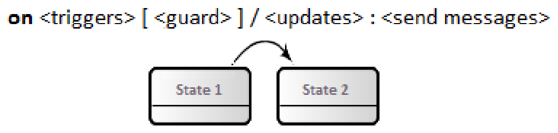


Figure 2: $\langle$HOE$\rangle^2$ Action Language

Gantt where T_j^i denotes the i^{th} task of iteration j. The same kind of planing can be declined for the development of the platform. As can be seen, the iterations form rectangles and squares in the table schedule: Each iteration starts from a consistant state and ends by a consistant state of the system. However, this kind of representation does not illustrate the refinements within a single iteration.

3.2 $\langle$HOE$\rangle^2$ – A New Action Language for HSM

The $\langle$HOE$\rangle^2$ language build on a number of concepts from Unified Modeling Language (UML) such as objects and associations [4, 15]. $\langle$HOE$\rangle^2$ Objects communicate through exchange of messages. Their behavior is captured by Harel's Statecharts [5, 15]. The $\langle$HOE$\rangle^2$ language proposes a number of extensions to UML:

- A new *Action language*. The $\langle$HOE$\rangle^2$ action language separates actions into two sequentially ordered parts: First the parallel updates of associations, followed by a parallel sending of messages – See Fig. 2.
- Using associations to define *Iteration domains* and *Indexes*. $\langle$HOE$\rangle^2$ introduces a syntax for the specification of parallel iteration domains for associations updates and sending actions.
- *Indexed regions*: Iteration domains can also be used to enumerate parallel regions. Each region is identified by its own index.
- *Transaction*: The concept of interface is extended with the ordering and direction of messages exchanged.

These extensions are detailled by Llopard et al. [4].

4 The CanHOE2 Tool

As we explained previously, CanHOE2 serves to support the $\langle$HOE$\rangle^2$ process and language. We choose to develop our tool with the several Eclipse technologies [16, 17, 18]. Eclipse offers a set of convenient user-interface components in the context of meta-modeling (Perspective Management, Common Navigator Framework, Ecore Standard, etc.). It provides some tools to easily display multieditors in the editor area, navigator and views. CanHOE2 can create models and diagrams associated to these models. Models are created using

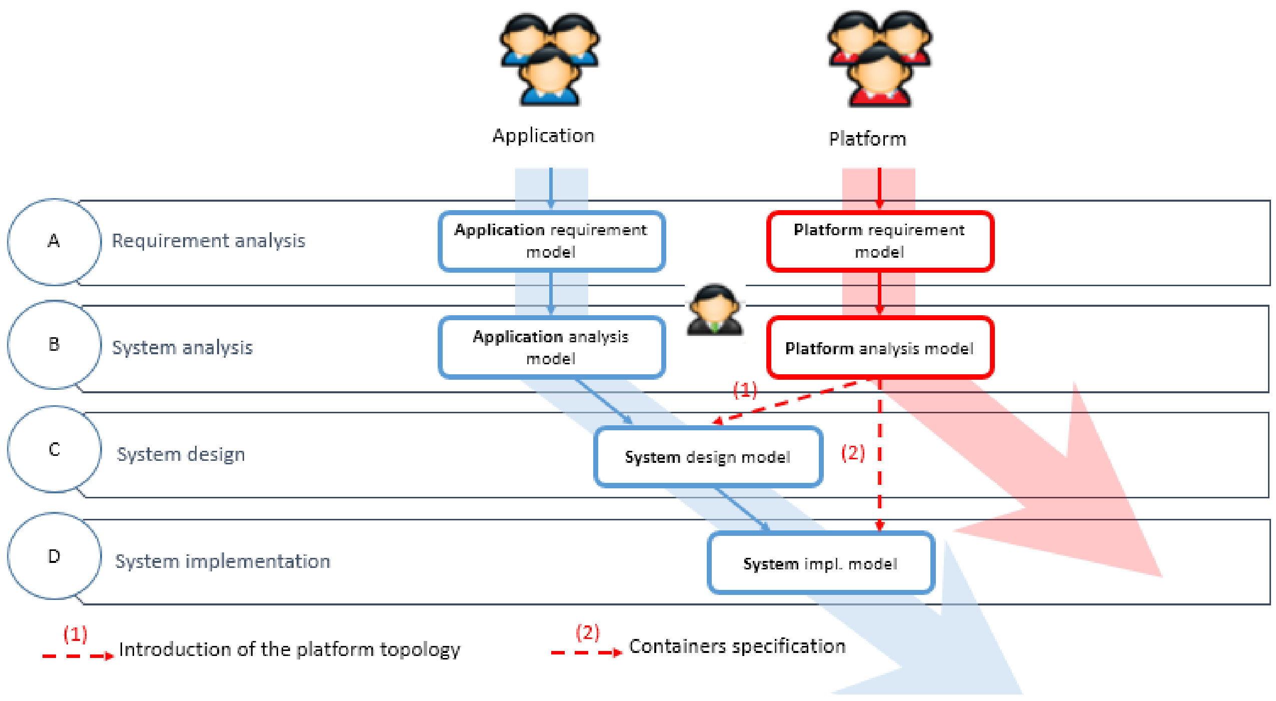


Figure 1: $\langle$HOE$\rangle^2$ a Collaborative Top-Down Dev. Process for Embedded System Design: Application and Platform Tracks

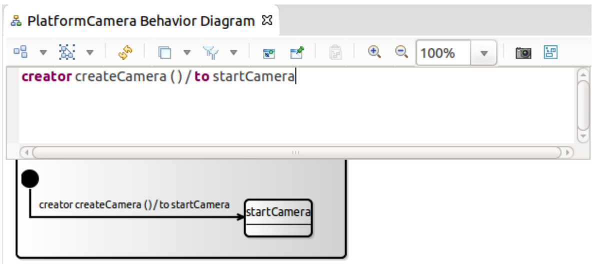
the Eclipse Modeling Framework (EMF) tool. It allows to instantiate models by providing a meta-model and save them in XML Metadata Interchange (XMI) markup language. It is suitable for metamodels and model exchanges. CanHOE2 also offers a textual editor for the $\langle$HOE$\rangle^2$ language. In this section, we detail the implementation choices, interface design and contributions of our tool.

4.1 Tool Design

CanHOE2 provides four editors corresponding to the four models described in the $\langle$HOE$\rangle^2$ method (see Fig. 1). Navigation in the models is based on the logical ordering of refinements within models. The project management tool supports consistent planning and definition of iterations obeying the model refinement dependencies. The iteration history is navigable. The user interface is split into three areas at a minimum (see Fig. 3):

- The *Navigator Area*, that allows the designer to intuitively navigate inside a streamlined hierarchy of $\langle$HOE$\rangle^2$ artefacts.
- The *Editor Area*: where the developer can write program using graphical elements or textual coding.
- The *View Area*, dedicated to project management and teamwork tools. It implements dedicated views to update information about the entities selected in the active editor.

As illustrated in Fig. 4, the developer may either write its program directly in $\langle$HOE$\rangle^2$ textual language or describe it graphically, assisted by a palette and a property panel: a palette is displayed and the developer is able to drag and drop entities from it. Information about graphical entities can be modified in the graphical editor, or in a property panel. Graphic and textual development can be complementary. Indeed, the developer can mix user friendly textual and graphics solutions, they will be automatically synchronized to each backup (see Fig. 4). By double-clicking on any graphical element, an embedded $\langle$HOE$\rangle^2$ language editor open up and let the programmer modify the actual code (see Fig. 5).



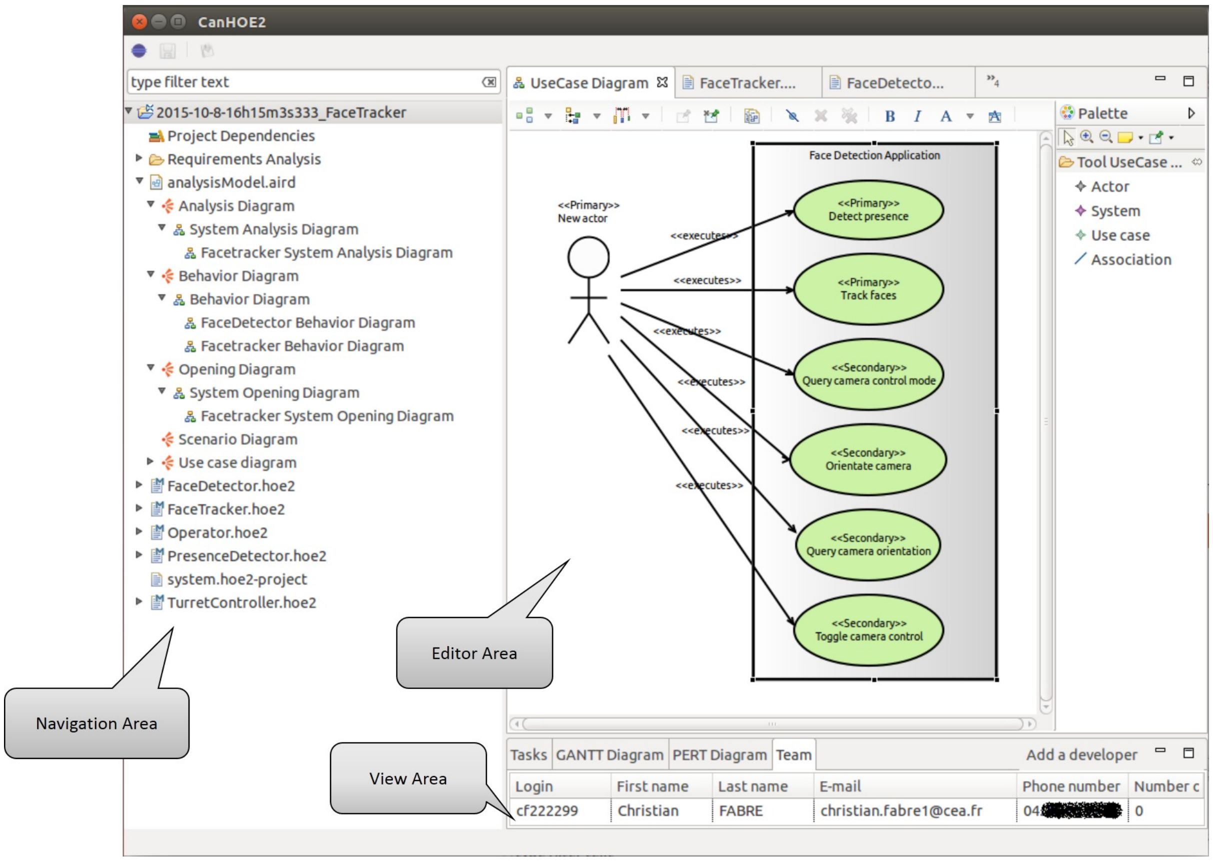
```
creator createCamera () / to startCamera
```

```
creator createCamera () / to startCamera
```

Figure 5: Embedded Editor for Transition

4.2 Tool Implementation

The first step is to derive a well-formatted Ecore metamodel from our canonical one. This step is fundamental since we need to consider the Ecore and its associated tool specificities. Once we have produced our



Login	First name	Last name	E-mail	Phone number	Number c
cf222299	Christian	FABRE	christian.fabre1@cea.fr	04[redacted]	0

Figure 3: The CanHOE2 Interface Dedicated to the ⟨HOE⟩^{2} Method and Language

Ecore metamodel, we defined the ⟨HOE⟩^{2} language syntax using Xtext/Xtend technology from the above defined Ecore model [17]. Xtext provides a set of tools for the implementation of domain-specific languages. From the ⟨HOE⟩^{2} metamodel, we derived a full concrete implementation of it. The compiler components of our language are independent of Eclipse or OSGi and can be used in any Java compliant environment. Xtext automatically generates the parser, a type-safe builder of our Abstract Syntax Tree (AST), the serializer and code formatter, the scoping framework and the text editor. The editor integrates syntax highlighting and error checking, among many other things. To achieve graphical models, we rely on the Sirius framework [18]. Sirius enables the visual design of complex systems (software, business activities, physics, etc.) and guarantees the consistency of the corresponding data (architecture, component properties, etc.).

4.3 Support Tools for Project Management in CanHOE2

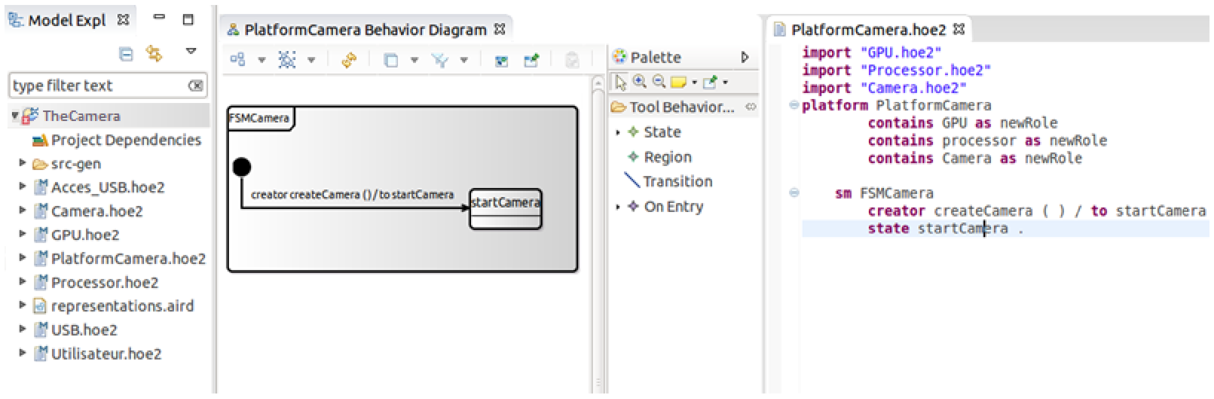
CanHOE2 offers several tools to support project management. Indeed, it can manage the authentification and collaborative work providing a global view on the progress of the project.

Authentication. In a regular work environment, an employee can only access the projects to which he has been assigned. In CanHOE2, once authenticated, the user accesses a personal dashboard that lists the projects in which he is involved. From this dashboard, he can also create new projects as a PM. According to his role in the project, one of two perspectives is opened: “Project Manager” or “Developer” perspective. A perspective is a window with adapted views to the user. Eclipse RCP can easily manage these perspectives and allow us to show a proper layout of the available views. The views in CanHOE2 are based on Java SWT [19].

The user authentication process goes through Lightweight Directory Access Protocol (LDAP). LDAP is a network protocol for accessing an electronic directory where you can reference the users (name, login, phone ...), machines or applications. Access to LDAP server is done via Java Naming and Directory Interface (JNDI) [20].

Collaborative development. Such a collaborative environment needs a database to capture all the artifacts manipulated and their evolution over time. CanHOE2 uses centralized Git repositories to this end that are accessed by CanHOE2 on behalf of each participant [21]. CanHOE2 uses two types of repositories:

- A configuration Git repository contains the infor-



```
import "GPU.hoe2"
import "Processor.hoe2"
import "Camera.hoe2"
platform PlatformCamera
        contains GPU as newRole
        contains processor as newRole
        contains Camera as newRole

    sm FSMCamera
        creator createCamera ( ) / to startCamera
        state startCamera .
```

Figure 4: CanHOE2 Synchronized Text and Graphical Editors

mation that CanHOE2 uses to connect to LDAP and the list of all CanHOE2 active repositories.

- Several project-specific Git repositories, where we store additional information about the project (task sheets) and resulting patterns of the CanHOE2 application (model use cases, scenarios).

Project monitoring. In order to help the project manager to monitor the project development and to plan different iterations on it, we implement traditional diagrams for project management such as GANTT and PERT diagrams. Internalized project management reduces development time and provides a monitoring interface to better manage the development team.

5 Case Study

In this section, we introduce a Face Tracker system. The face tracker system consists of application for face detection, which is hosted by an Oriented Camera platform. This platform includes a general purpose processing unit, a Passive InfraRed (PIR) sensor and a bracket on which the camera is attached. Two servomotors orientate the bracket for pan and tilt.

Fig. 3 illustrates use cases diagram of the Face Tracker application . Tables 2 and 3 show multiple use cases and scenarios, respectively, that formalizes the requirements. From these use cases and scenarios, we can now define a task list and its corresponding order of execution. We present hereafter a non-exhaustive list of tasks that have to be performed:

- T^1: Refinement of UC 1 & 2
- T^2: Initiation of the system analysis
- T^3: Refinement of UC 3, 4, 5 & 6
- T^4: Refinement of the system analyis for UC 3 & 4
- T^5: Refinement of the system analyis for UC 5 & 6
- T^6: Initiation of the system design

Table 2: Usecases of the Face Detection Application

Id	**Causality**	**Name**
Description		
1	Primary	Detect presence
The actor wants to know when somebody enter the monitored zone.		
2	Primary	Track faces
The actor wants to track faces of people entering the monitored zone.		
3	Secondary	Toggle camera control
The actor wants to switch between manual and automatic tracking modes		
4	Secondary	Query camera control mode
The actor wants to know the current tracking mode		
5	Secondary	Orientate camera
The actor set the camera's orientation.		
6	Secondary	Query camera orientation
The actor wants to know the camera's curent orientation.		

- T^7: Refinement of the system design
- T^8: System implementation

Once this list is completed, we can define the execution priority of tasks and hence the list of our different iterations.

- $I_1 = T^1 + T^2$: Refinement of the use cases 1 & 2 and system analysis initiation
- $I_2 = T^3 + T^4 + T^5$: Refinement of requirements and system analysis
- $I_3 = T^6 + T^7$: System design
- $I_4 = T^8$: System implementation

Table 1 illustrates our iterations planning. In what follows, we will focus on I_1 and I_3 iterations. However, before the beginning of the design phase (i.e. iteration I_3), the platform development team have to start to work on it. We will call that iteration I_1'. We named our platform, "The Oriented Camera platform".

Table 3: Scenarios of the Face Detection Application

Id	Nature	Name / Description
1-1	Nominal	Presence detected notification
		The actor subscribes for updates of presence in the monitored zone. The application notifies to the user when somebody enters or exit the monitored zone. The actor unsubscribes when he does not want updates anymore.
2-1	Nominal	Face tracking notification
		The actor subscribe to people's faces and the position in the monitored zone. The application notifies to the user with the position of a face in the monitored zone every 2 *s*. The actor unsubscribes when he does not want to track people faces anymore.
3-1	Nominal	Switching to automatic mode
		The system is in manual mode. The user toggles it to automatic mode.
3-2	Nominal	Switching to manual mode
		The system is in automatic mode. The user toggles it to manual mode.
4-1	Nominal	Querying control mode
		The actor asks for the camera's control mode. The system answers wether the camera is in automatic or manual mode.
5-1	Nominal	Manually orienting the camera
		The system is in manual mode. The user wants to move the camera relative to the curent position. The camera is moved by the requested *pan* and *tilt* angles.
5-2	Error	Manually orienting while in auto. mode
		The system is in automatic mode. The user wants to move the camera relative to the curent position. The system refuses because the camera is in automatic mode.
5-3	Error	Manually orienting the cam. hits a stop
		The system is in manual mode. The user wants to move the camera relative to the curent position. The camera is moved by less than the requested *pan* and *tilt* angles as it hits a hard stop.
6-1	Nominal	Camera position query
		The actor asks the system about the camera position. The system answers with the position.

Iteration I_1: Refinement of the use cases 1&2 and system analysis initiation

During this iteration, we can detail the use cases identified but also initiate the analysis phase. Fig. 6 illustrates an analysis model of the Face Tracker system : the system is composed by three objects (1) a presence detector for detecting a presence in a specific area, (2) a face detector for detecting a face and its position from a video stream and (3) a turret controller for controlling the orientation of a camera a controllable turret two angles.

In this iteration we have also to model the behavior of our application, and its response to exchanged messages (See Fig. 7).

Iteration I_1': Requirements and system analysis of the platform

Tables 4 and 5 gives a list of use cases and scenarios of the platform. These lists can be completed and refined with theirs corresponding diagrams.

To model the platform, we use the same diagram for

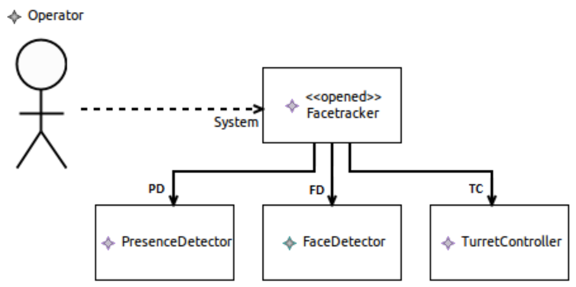


Figure 6: Face Tracker system opening diagram

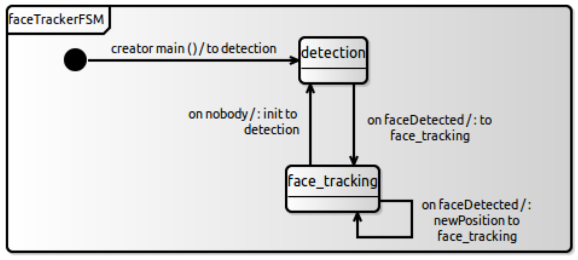

faceTrackerFSM
creator main () / to detection
detection
on nobody / : init to detection
on faceDetected / : to face_tracking
face_tracking
on faceDetected / : newPosition to face_tracking

Figure 7: Face Tracker system behavior diagram

modeling the application with a particular notation for the different elements of the platform. Fig. 8 shows the analysis model of the platform. This platform allows the hosted application to control, via two engines, the position of the camera by orientating it around two axes. The model is composed by two worlds a Processor and Microcontroller. They are provided with resources in terms of computing and memory, and communication. They also provide different containers allowing access to the communication resources and to the peripherals of the platform.

The technical choices for the platform design are:

- The Raspberry PI platform: It is adapted to the field of image processing and allows installing and uninstalling Python programs running one program at a time. It allows also the update of time and date settings.
- The turret control platform: This platform provides the use cases (1) subscribtion to presence detection, and (2) turret orientation. The turret can be used to support the sensors or actuators, such as a camera, an ultrasonic radar, a projector, etc.
- The Arduino UNO platform: It is a prototyping platform for the control of sensors and, digital or analog actuators.

From these three platforms we have designed the composition diagram (see Fig. 9). From this level of modeling we can start the system design.

Iteration I_3: System design

Table 4: Use cases of the Oriented Camera platform

Id	Causality	Name
Description		
1	Primary	Install Firmware
The actor installs a new firmware in the platform.		
2	Primary	Toggle the application on/off
The actor turn on/off the application.		
3	Primary	Time & date settings update
The actor sets the time and date settings.		
4	Secondary	Time & date settings query
The actor gets the time and date settings.		
5	Secondary	Firmware version query
The actor gets the firmware version.		

Table 5: Scenarios of the Oriented Camera platform

Id	Nature	Name
Description		
1-1	Nominal	Firmware installed
The actor installes a new firmware.		
1-2	Error	Firmware installation error
A firmware is already installed on the platform. The actor wants to install the same version or an older version of the firmware. The firmware cannot be installed.		
2-1	Nominal	Toggle the application on/off
The actor turn on the application. The application runs till the actor decides to turn off the application. The application is switched off.		
3-1	Nominal	Time & date settings updated
The actor sets the time & date. The platform reset its clock to the entered time & date.		
4-1	Nominal	Time & date settings query
The actor asks the system about the time & date. The system answers with its internal time & date.		
5-1	Nominal	Firmware version query
The actor asks the system about the firmware. The system answers with the firmware version.		

The I_3 iteration initializes and starts the system design phase satisfying the separation of objects, into the two worlds of the platform (See Fig. 10).

6 Conclusion & Perspectives

The ⟨HOE⟩^{2} method defines a single process used to develop both applications and platforms. It defines precisely what information is sent from the platform model to the application model during development. Its companion language suports parallelism and is amendable to polyhedral analyses. It is made of (1) a new action language limited to association updates and sending of messages, (2) domains, iterators and indexes based on associations.

In this paper, we presented CanHOE2, a modelling and process management tool for the ⟨HOE⟩^{2} method and its language. Its main contribution is that not only does it supports the ⟨HOE⟩^{2} language, as any modelling tool, but also the associated process. The

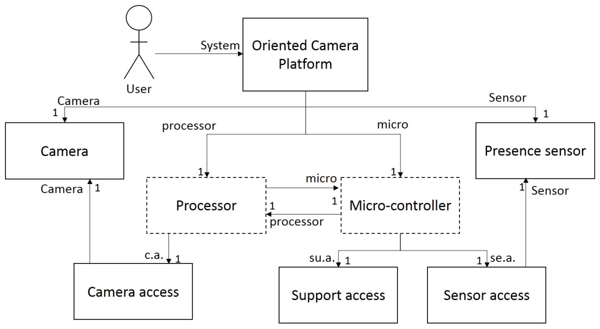


Figure 8: Oriented Camera Platform Analysis

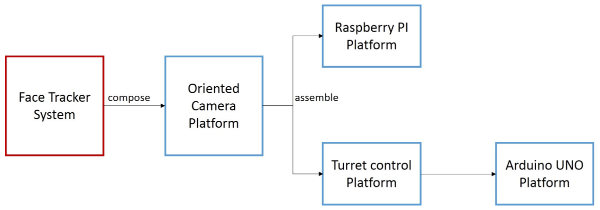


Figure 9: Oriented Camera Platform Composition

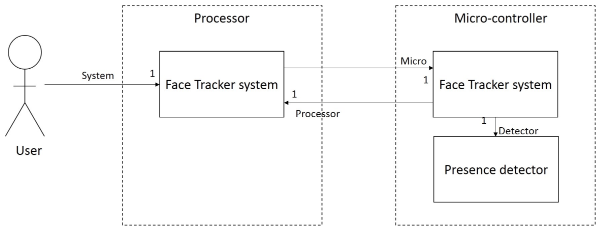


Figure 10: Face Tracker system design

support of management and cooperation is deeply integrated with support of modeling.

In future work, we want to extend CanHOE2 with more support for the method, in particular towards the models simulation and ⟨HOE⟩^{2} model-to-model transformations.

Aknowledgements: This work was partially funded by the Artemis-JU and the French "*Ministère de l'économie, de l'industrie et du numérique*" for the Artemis project COPCAMS, GA 332 913, `http://copcams.eu`.

References

[1] Nicolas Hili, Christian Fabre, Sophie Dupuy-Chessa, and Stéphane Malfoy. "Efficient Embedded System Development: A Workbench for an Integrated Methodology". In: *Proc. of the 6th Embedded Real-Time Software and Systems Congress (ERTS^2 2012)*. Toulouse, France, Feb. 1, 2012. URL: `http://hal.inria.fr/hal-00671966/`.

[2] Nicolas Hili, Christian Fabre, Sophie Dupuy-Chessa, and Dominique Rieu. "A Model-Driven Approach for Embedded System Prototyping and Design". In: *Proc. of The IEEE International Symposium on Rapid System Prototyping (RSP 2014)*. New Delhi, India, Oct. 16–17, 2014. DOI: `10.1109/RSP.2014.6966688`.

[3] Nicolas Hili, Chrisitan Fabre, Ivan Llopard, Sophie Dupuy-Chessa, and Dominique Rieu. "Model-Based Platform Composition for Embedded System Design". In: *Proc. of IEEE 8th International Symposium on Embedded Multicore Many-core Systems-on-Chip (MCSoC-14)*. University of Aizu, Japan, Sept. 23–25, 2014. DOI: `10.1109/MCSoC.2014.31`.

[4] Ivan Llopard, Albert Cohen, Christian Fabre, and Nicolas Hili. "A Parallel Action Language for Embedded Applications and Its Compilation Flow". In: *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*. SCOPES '14. Sankt Goar, Germany: ACM, 2014, pp. 118–127. ISBN: 978-1-4503-2941-5. DOI: `10.1145/2609248.2609257`.

[5] David Harel. "Statecharts: a Visual Formalism for Complex Systems". In: *Science of Computer Programming* 8.3 (1987), pp. 231–274. ISSN: 0167-6423.

[6] Sébastien Gérard. "Modélisation UML exécutable pour les systèmes embarqués de l'automobile." PhD thesis. Université d'Evry, Oct. 2000.

[7] Denis Aulagnier et al. "SoC/SoPC development using MDD and MARTE profile". Anglais. In: *Model Driven Engineering for Distributed Real-time Embedded Systems*. Ed. by Jean-Philippe Babau et al. ISTE, Mar. 2009.

[8] A. Basu et al. "Rigorous Component-Based System Design Using the BIP Framework". In: *Software, IEEE* 28.3 (Apr. 2011), pp. 41–48. ISSN: 0740-7459. DOI: `10.1109/MS.2011.27`.

[9] IBM Rational. *Rhapsody*. URL: `http://www.ibm.com/software/products/en/ratirhap` (visited on 2015-10-13).

[10] IBM Rational. *Rules Composer*. URL: `http://www-01.ibm.com/support/docview.wss?uid=swg27019384` (visited on 2015-10-13).

[11] Sébastien Gérard, Cédric Dumoulin nad Patrick Tessier, and Bran Selic. "Papyrus: A UML2 tool for Domain-Specific Language Modeling". In: vol. 6100. Springer Verlag, 2010.

[12] Sébastien Gérard. *On the Papyrus' USE: Usage, Specialization and Extension*. Sept. 6, 2010. URL: `http://www.eclipse.org/modeling/mdt/papyrus/usersTutorials/resources/TutorialOnPapyrusUSE_d20101001.pdf`.

[13] Eclipsepedia. *Papyrus Qompass*. June 5, 2014. URL: `https://wiki.eclipse.org/Papyrus%5C_Qompass`.

[14] Eclipsepedia. *Papyrus/Codegen/Cpp description*. Sept. 29, 2015. URL: `https://wiki.eclipse.org/Papyrus/Codegen/Cpp%5C_description`.

[15] Object Management Group. *OMG Systems Modeling Language (OMG SysML), Version 1.3*. Object Management Group, 2012.

[16] *Eclipse*. The Eclipse Foundation. URL: `http://www.eclipse.org` (visited on 2015-10-13).

[17] *Xtext: a Framework for Development of Programming Languages and Domain Specific Languages*. Eclipse Foundation. URL: `http://www.eclipse.org/Xtext` (visited on 2015-10-13).

[18] *Sirius: an Eclipse Project which Allows the Creation of Graphical Modeling Workbench by Leveraging the Eclipse Modeling Technologies*. Eclipse Foundation. URL: `http://www.eclipse.org/sirius` (visited on 2015-10-13).

[19] *SWT : an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented*. Eclipse Foundation. URL: `www.eclipse.org/swt/` (visited on 2015-10-27).

[20] *Java Naming and Directory Interface*. Oracle. URL: `http://www.oracle.com/technetwork/java/jndi/index.html` (visited on 2015-10-13).

[21] *Git: a Free and Open Source Distributed Version Control System*. git. URL: `http://git-scm.com` (visited on 2015-10-13).

Session 26

Domain Specific Language 2

Friday 29th, 09:00 – Guillaumet

The Unified Model-Based Design: how not to choose between Scade[1] and Simulink[2].

Jean-Louis Dufour, Bertrand Corruble, Bertrand Tavernier.

Sagem Défense Sécurité, groupe SAFRAN.

Summary: software projects using Simulink or Scade use in fact a subset of Simulink or Scade. The 'alignment' of these two subsets gives rise to a new concept, the 'Unified MBD', interesting in two respects: on the academic side, it gives a simple semantics to our subset of Simulink, and on the industrial side, it permits at almost no cost the double-skill Scade/Simulink for software and system engineers. For the data-flow part, the unified subset is simply the **'clockless'** part of the Scade/Simulink intersection. For the control-flow part, the unified subset is much more restricted, but the fundamental aspect is that it strictly conforms to a well-known paradigm called **'mode-automatas'**.

Keywords: Model-Based Software Engineering, Languages and Compilers, Scade, Simulink.

1. The 'Unified Model-Based Design' concept

Model-Based Design ('MBD') is today a major paradigm in the engineering of critical embedded software. Here we will focus on implementation models, from which code is automatically generated. The main actor is Simulink, but for certified systems (avionics [our context is DO178 DAL A], railway, nuclear), it has to share the cake with Scade. The simplest (but fuzzy) definition of the subject of this paper, the 'Unified MBD', is the intersection of Scade and Simulink. It is an answer to two problems.

The first and principal problem is an industrial and human one. When the R&D team cannot choose between the two tools, this has unpleasant consequences. Here are the ones we want to address:

- It potentially doubles training costs, and developers have a 'cold start' at each language commutation,
- the choice of the 'right' language for in-house libraries can be Cornelian, because it dictates the language for the related products.

The second design driver is simply the main limitation of each tool (in the narrow context of software engineering):

- Simulink's DNA is development, certification was handled only recently: the Simulink subset we use has no formal semantics, contrary to Scade, and the tentative formal semantics are too sophisticated to be of any use in an industrial context ([HamonRushby2007]),
- Scade's DNA is certification, ease of development is perfectible: the Scade simulation environment is suitable for unit debugging, but integration and functional debugging is better done out of the tool, contrary to the seamless environment of Simulink.

This very last point is one of the two keys of Unified MBD: our wish list to Scade contains today mainly tooling points, because **the Scade language has reached a good expressiveness level, almost sufficient to deal with most critical applications**. To be completely clear, when we look at the C source of the application part of our critical avionics software currently under Model-Based development (the other parts are the Operating System and the Boot):

[1] ANSYS, Inc.
[2] The MathWorks, Inc.

- A few percent come from manual coding, because datatypes are low-level (bitfield manipulations) or algorithms are low-level (sin, sqrt) or sequential (sorting an array, Gram-Schmidt orthogonalization),
- a few percent come from automated coding of state-machines (because our applications contains few moding and are mainly algorithmic; on cockpit or in railway, it could be very different),
- and everything else comes from automated coding of dataflow diagrams.

Independently of this dataflow/state-machine ratio (which is application-dependent), we make a different use of the dataflow features and of the state-machine features:

- almost all dataflow constructions are used: they are very similar in Scade and Simulink, are very intuitive (well readable by system engineers, this is a key point for peer reviews). This will be illustrated in the next section;
- almost none of the state-machine constructions are used: their semantics is often subtle and slightly different between Scade and Simulink, so this is directly conflicting with the first goal of Model-Based Design which is communication between engineers. This will be illustrated in the 3rd section.

Let's call 'Simple Scade' the part of Scade mentioned above (almost complete dataflow + almost nothing of state-machines). The second of the two keys of Unified MBD is that **Simple Scade is syntactically included into Simulink**. What does it mean? It means that without any semantic analysis, you can structurally translate (almost box to box and wire to wire) a Scade diagram into an 'equivalent' Simulink diagram. Equivalent is put in quotation marks because without a formal semantics for Simulink, it cannot be proved, but it can be verified by test (as in [Caspi&Coll2003]) or by expert judgement.

Let's call 'Simple Simulink' the image of Simple Scade under the syntactic translation. Contrary to Simple Scade, which is a syntactic subset of Scade, checking that a Simulink model belongs to the Simple subset requires a semantic analysis (simple: the transposition of Scade typechecking into Simulink). But once you are in the Simple Simulink subset, you can again syntactically translate towards Simple Scade.

Now we can give the **definition of the Unified MBD: it is the 'pack' formed of Simple Scade, Simple Simulink and their two syntactic translations**. The two translations have been prototyped, to check the robustness of the concept (the translators are not qualified, and we don't plan to reuse certification credits). This concept can be seen as a 'simple industrial version' of the academic works [Caspi&Coll2003] and [Scaife&Coll2004]. The knowledge of what is 'useful' was the key to keep things simple, and on this subset we claim to have the first 'truly operational' semantics of Simulink, usable in an engineering context.

To be completely clear, we must emphasize two points:

- The translators are really just proofs of concepts, and are far from being industrial tools. They have been quickly developed in MATLAB[3] to take advantage of its Simulink API for reading / writing / checking models (Scade models are stored in XML, for which MATLAB has also an API).
- The Unified MBD is not an Esperanto (i.e. a 3rd modeling language): it is more like a duo 'British Basic English[4]' / 'American Basic English', and the translators mentioned above simply change some 's' into 'z' or vice-versa. Projects still use Scade or Simulink, and the Unified MBD only appears as modeling rules. In this, it can be opposed for example to Synoptic

[3] The MathWorks, inc.
[4] https://en.wikipedia.org/wiki/Basic_English

[Cortier&Coll2010], which proposes a new modeling language, but it cannot be opposed to the P project [PothonBordin2013], because the Pivot language is not a language for modeling but for communicating models between tools.

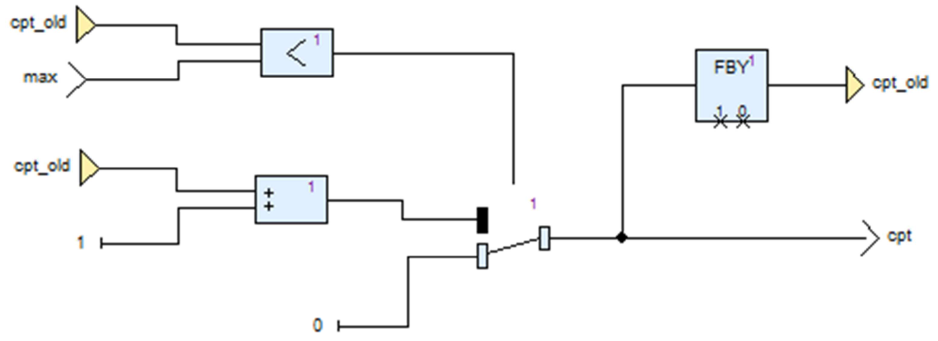
2. The dataflow part

Scade is not a pure data-flow language: it is a 'synchronous' data-flow language. It means that computation is not only driven by the values of the signals, but also (and firstly) by their 'clocks'. It permits compilation (pure data-flow is not really compilable, excepted in asynchronous hardware). In practice, clocks are not used, even for multi-tasking applications, and when you need to suspend an operator, you 'conditionally activate it on a value' and not on a clock: this is illustrated in the next sub-section. The reason is that it is conceptually simpler for engineers (the generated code is often the same), this is to be contrasted with the hundreds of academic papers on clock-calculi that have been published in the thirty past years. This is the main characteristic of the unified data-flow: to be **clockless** (technically speaking, we should say 'single clock', but clockless is more meaningful).

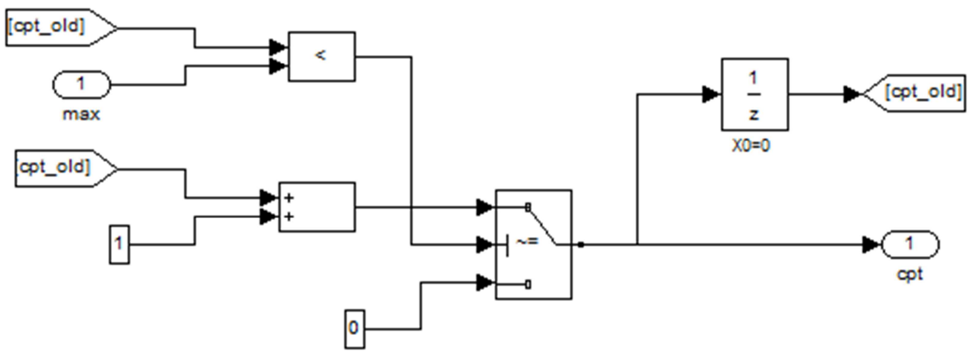
This section is structured according to the well-known equation 'Algorithms + Data Structures = Programs'.

2.1 Algorithms

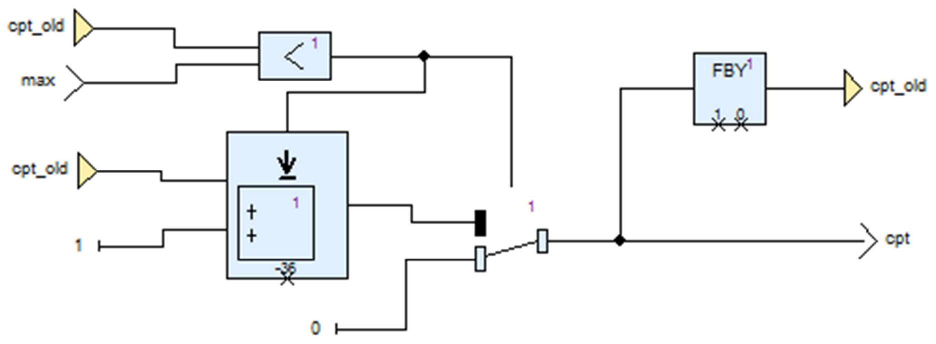
The syntactic translations are in general completely obvious (this is the very reason why the Unified concept makes sense). Here is a standard cyclic counter in Scade



and its translation in Simulink:

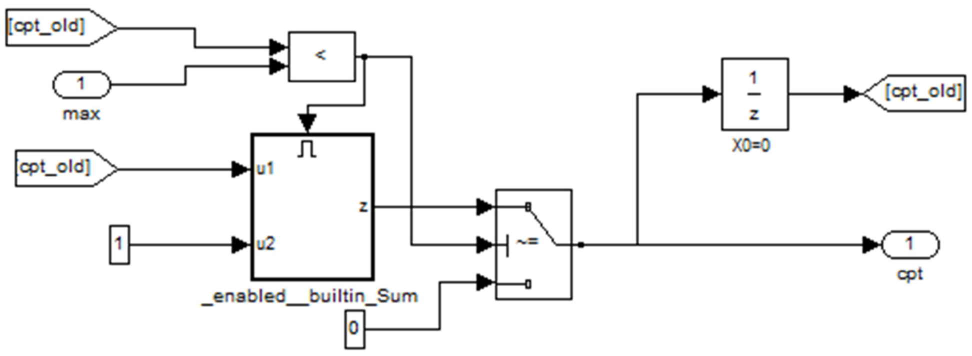


The not-completely-obvious aspects are with what Scade calls 'higher order patterns', typically the conditional activation or the map: in Scade they operate on a sub-block, whereas in Simulink they are in the sub-block (they 'qualify' it). Following the MBD motto (and long before, a famous French statesman), a small drawing is better than a long speech: here is the same counter, but with a conditional addition (to explicitly avoid overflow).

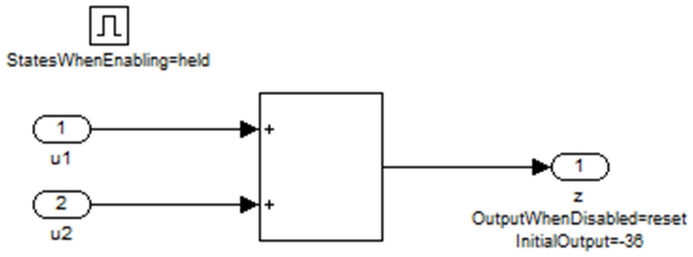


The strange block with a downwards arrow means ‘either activate the sub-block ‘+’, or return -36’. The default value -36 is part of the top-level block and not of the sub-block. The sub-block is the standard ‘+’, which doesn’t care about conditional activation or default result.

The Simulink translation looks identical, but in fact it is not:



The sub-block ‘_enabled__builtin_Sum’ is not the standard ‘+’: it contains an ‘Enable Port’ block and the default value -36 as a parameter of the output port ‘z’:



The translation can be improved by using the ‘block parameters’ mechanism of Simulink: this way, the ‘-36’ will appear in the top-level block, but there still will be a supplementary hierarchical level compared to Scade. The translation is syntactical (and reversible), but the structure is slightly changed.

2.2 Data structures

The only subtlety is the definition of vectors and arrays.

On the one hand, Scade, like C or Ada, is completely rigorous about that:

- a scalar is not a vector of dimension 1,
- a matrix is a vector of vectors,
- there is a unique 'vector constructor', which construct vector from scalars, matrices from vectors, etc… (same simplicity for 'vector projector'),
- the dimension of a vector is explicitly declared, and to permit generic algorithms (dot product for example, for any dimension), symbolic dimensional parameters 'M', 'N', etc. can be used, with a subset of arithmetic. For example, the concatenation of two vectors of size 'M' and 'N' is a vector of size 'M+N'.

On the other hand, Simulink, like Matlab, tries to be user-friendly, but to make things more 'interesting', it is not user-friendly in exactly the same way that Matlab is:

- a scalar is a vector of dimension 1 (in Matlab, it is a 1*1 matrix because vectors don't exist [row and column matrices do the job]), and is often equivalent to a 1*1 matrix,
- a matrix is not a vector of vectors,
- you cannot use exactly the same built-in blocks (or the same configuration of …) to construct/project vectors and matrices,
- the dimension of a vector or a matrix need not to be explicitly declared (even if it has to be statically known for code generation purposes).

The first consequence is that the Scade→Simulink translation is sometimes dimensionality-dependent, but hopefully this information is contained in the Scade diagrams: the translation is still syntactic.

The second consequence is more critical: for the moment, generic vector/matrix algorithms cannot be syntactically translated from Simulink to Scade, we can only syntactically translate each instance separately. Type-checking vector/matrix generic algorithms is a complex problem (see [JoishaBanerjee2006] for MATLAB), but hopefully, academic research has always kept this subject on its radar screen. Historically, it had purely academic motivations like APL evolutions or compiler efficiency motivations for FORTRAN, but today there is a new boost driven by parallelism/many-cores, and the concept of 'shape and dimension polymorphism' is increasingly studied. Concerning our Simulink→Scade translation problem, the type inference mechanism implemented in the 'Repa' package of Haskell ('Regular Parallel arrays') is not far from our needs [Keller&Coll2010].

3. The state-machine part

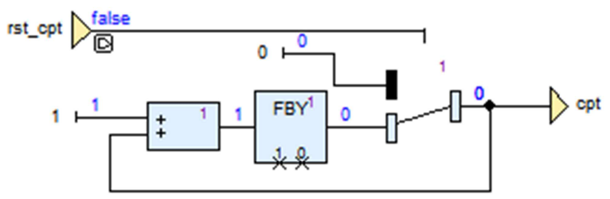
In term of 'Scade-Simulink intersection', state-machines are the exact opposite of the dataflow part: almost nothing can be unified (if you want to ensure the same look-and-feel and simple translations). In fact, as soon as you introduce hierarchy, the semantics differ. To try a simple explanation, in Simulink a sub-automaton is just a drawing artifact, whereas in Scade, a sub-automaton is a true state-machine.

Hopefully, the state-machines we use are ridiculously simple compared to the expressiveness of both state-machine formalisms, and the small intersection (non-hierarchical) makes sense. We have interesting use-cases of (one-level) hierarchy, so the subject is not closed, but for the moment the unified subsets are non-hierarchical.

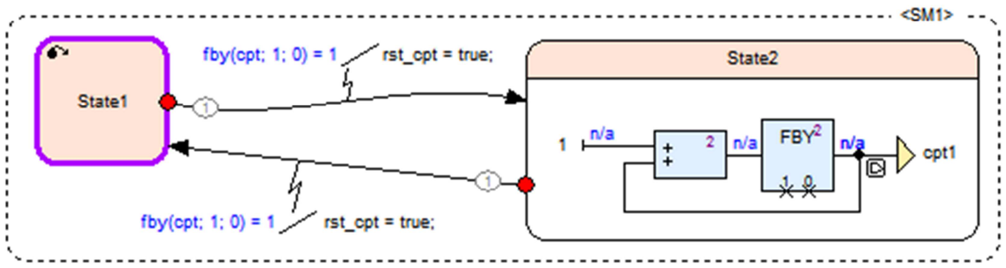
3.1 The SCADE 'Unified Modeling Style'

Scade designers have pushed the dataflow/state-machine integration to the extreme, we will illustrate this on the following operator which generates two integers 'cpt' and 'cpt1' such that:

If cpt1>=0 then cpt = cpt1



default values :
rst_cpt = false
cpt1 = -1



The state-machine <SM1> is not an operator (the operator is the whole figure): it is just an equation (i.e. a definition of variables), at the same level than the dataflow definition of 'cpt'. It defines the two variables 'rst_cpt' and 'cpt1', and what is interesting is that 'cpt1' is defined by a dataflow INSIDE 'State2'. In fact, 'State2' is an operator, working 'part time' (in the figure it is at rest, hence the 'n/a's), which could contain itself another state-machine, etc.

The underlying paradigm is extremely clean and elegant: instead of 'state-machine' (a dataflow containing a FBY [i.e. an 1/z] is also a state-machine, after all) we should speak of 'mode-machine' (or 'variant-machine', 'schizophrenic-machine', 'Janus-machine', etc.). The difference is that a state is just a value, whereas a mode is a behavior: a mode is simply an operator implementing one particular personality of many. The first data-flow embodiment of this paradigm were the 'mode-automatas' of [MaraninchiRémond1998].

The Scade implementation of this paradigm (see [Colaço&Coll2005] or the Scade Suite documentation) is rightly called the 'Unified Modeling Style' (not to be confused with our 'Unified Model-Based Design'). And as many implementations, it is (substantially) more complex than its specification: to give an idea, let's just say that none of the three authors has ever reached the end of the Scade Language 'Tutorial' [Esterel2011], which models mainly with state-machines a descendant of Gérard Berry's stopwatch (published in [Berry1991], see also [Halbwachs1993] ; itself a descendant of Harel's wristwatch [Harel84]).

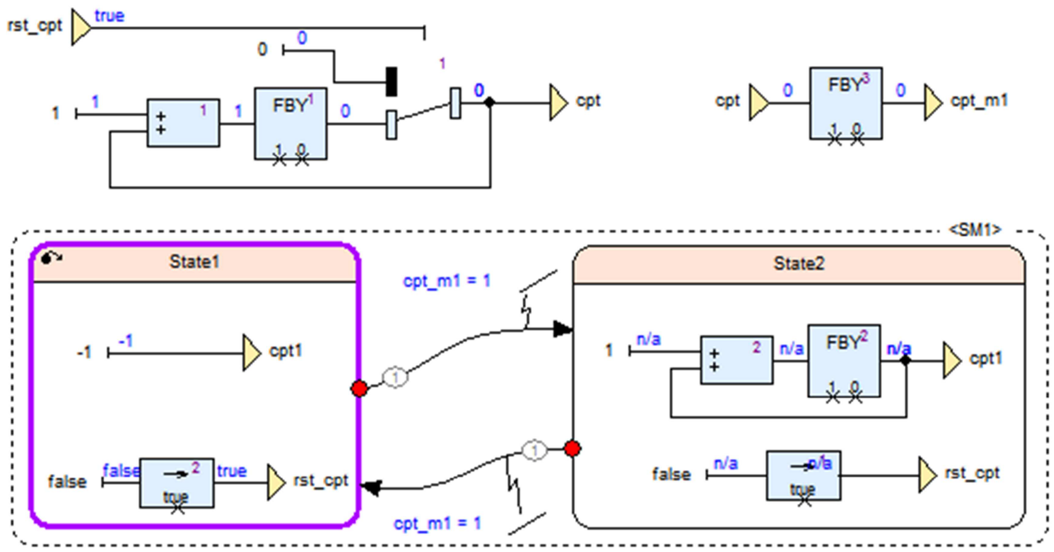
It would be too long to exhaustively enumerate all the limitations we impose on state-machines: weak/synchro/resume transitions, signals, 'last' operator, etc. In fact almost everything has been forbidden, and it is easier to explain what we kept. Let's call a state-machine 'simple' when

- the states are independent operators (no 'last': no sharing between states), restarted (and not resumed: inner states are re-initialized) at each entering transition, and contain only (a subset of) built-in operators,
- transitions are 'strong', i.e. their conditions are the first thing computed, then the new state is determined and executed,
- transitions have no actions, no events and combinatorial conditions (no temporal operator).

So, this is a minimalist interpretation of the paradigm. To transpose the former operator into the unified subset, we have to:

- replace the default value mechanism on 'cpt1' and 'rst_cpt' by an explicit definition in every state,

- replace the two actions on transition by adequate initializations in the modes (transition actions are pushed inside the modes),
- replace the 'fby' in the conditions by the elaboration of a new variable 'cpt_m1' (the 'fby' is pushed outside the state-machine).



This last paragraph is a bit technical and can be skipped; it is just an illustration of the countless subtleties that occur with state-machines, which we consider to be inacceptable in a safety-critical design. If you observe only the outputs, the two previous diagrams are indistinguishable. But if you observe also the inner signal 'rst_cpt', there is a difference: it is false in the first diagram and true in the second (look at the upper-left corner of the diagrams). It occurs only during the first cycle (the initial start or 'cold start'), and it is the reason why the first diagram is not accepted by our Scade→Simulink translator (the 2nd diagram is OK). More precisely, <SM1> in the first diagram is a state-machine, but is not a 'mode-machine', because 'State1' is not an operator: it behaves differently in the initial start (rst_cst = false) and in a warm start (a restart in the middle of a run, coming from State2; in this case rst_cst = true). In other words, State1 has an 'extra-sensory' knowledge (not given by its inputs and state) of the fact that a start is an initial one or a warm one. Again in other words, if we had to translate this in the unified subset, we would have to add a supplementary input to State1 to give him this information. We consider that it is no more a syntactical translation, but more fundamentally we consider that such a subtle semantic point is not acceptable in a safety-critical design.

3.2 Simulink: classicism

State-machines in Simulink are built with a toolbox called 'Stateflow'. In terms of expression power and complexity, they are comparable to Scade, but they are 'classic' w.r.t. Harel's 'seminal synthesis' [Harel1984]: the behavior of the states is sequential code.

The search for semantic simplicity and the intersection with Scade have led to a very reduced subset, for example from the most general form of transition

'event_trigger[condition]{condition_action}/transition_action'

we keep only

'[condition]/transition_action'

with global constraints on the transition actions in order to ensure the 'mode-machine' aspect.

3.3 The representation of the intersection

Concerning the dataflow part, the specification of the translations is very simple, from ‘drawing’ to ‘drawing’, and the implementation is just as easy, from ‘syntax tree’ to ‘syntax tree’.

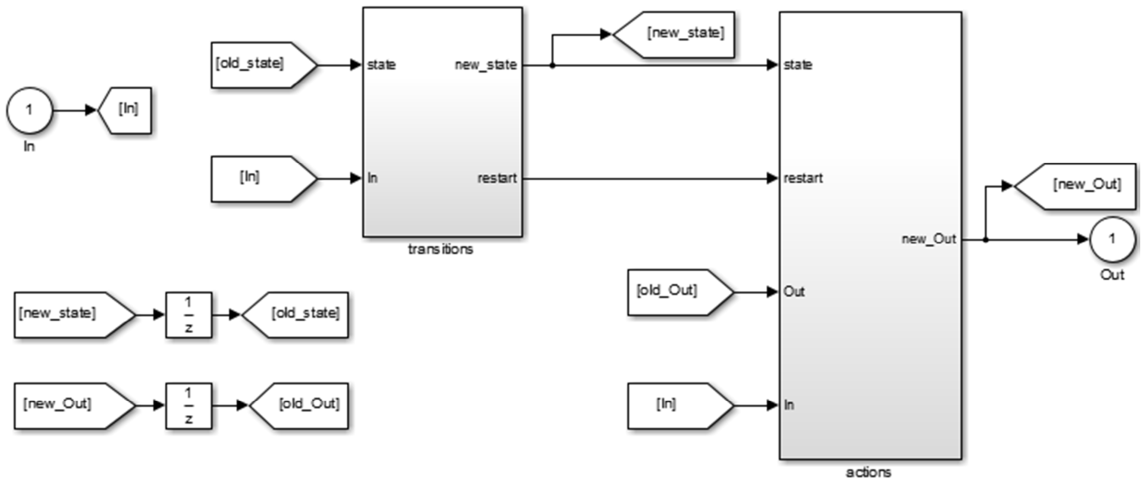
Concerning the state-machine part, things are different, and the need for an intermediate language was felt very quickly. The classical approach is to design a third language, and we could have used the ‘Pivot’ language of the ‘P’ research project of which we are partner ([PothonBordin2013]). But the aims of the P-project and of the Unified MBD are not the same, which explains why we have adopted another solution:

- the P-project needs a pivot language for ensuring semantic correctness of transformations of models, nobody will never directly read or write P models,
- the Unified MBD needs an operational, but also simple and readable way of specifying (for human end-users) subsets of Scade or Simulink.

Our solution for the specification (and translation) of unified state-machines is a dataflow design-pattern, consisting of (see also next figure):

- a set of states (left-bottom part of the diagram),
- a pure (stateless) function ‘transitions’ computing the next state and the restart order (middle of the diagram),
- a pure (stateless) function ‘actions’ using this next state and restart order to compute the output(s) (right of the diagram).

So, when we want to incorporate a new feature of, say, Stateflow, into the Unified MBD, we don’t have to check the simplicity of translation in Scade: we check the simplicity of translation in this design-pattern. This is not a third language; this is a pivot design pattern inside the Unified dataflow: formal semantics of state-machines comes ‘for free’.



4. The Standard Library, and conclusion

A common in-house ‘standard’ library has been defined, with basic operators like matrix computations and filters. The Scade and Simulink version share their specification and their test-cases. This way, we claim to have unified the look-and-feel of our models and to permit at almost no cost the double-skill Scade/Simulink.

The goal of this paper is to show that for critical software, the interesting subset of Scade/Simulink is not so big than that. But of course, the way forward is to continue to extend the two subsets, according to use-cases needed by development projects.

5. References

[Berry1991] Gérard Berry. Programming a digital watch in Esterel v3. TR 08/91, Centre de Mathématiques Appliquées, Ecole des Mines de Paris, 1991.

[Caspi&Coll2003] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating Discrete-Time Simulink to Lustre. International Conference on Embedded Software (EMSOFT), 2003.

[Colaço&Coll2005] J-L Colaço, B.Pagano and M. Pouzet. A conservative extension of synchronous data-flow with state machines.EMSOFT'05.

[Cortier&Coll2010] A. Cortier, L. Besnard, J.P. Bodeveix, J. Buisson, F. Dagnat, M. Filali, G. Garcia, J. Ouy, M. Pantel, A. Rugina, M. Strecker, and J.P. Talpin. Synoptic: A Domain-Specific Modeling Language for Space On-board Application Software. Chapter 3 of the book edited by S.K. Shukla and J.-P. Talpin 'Synthesis of Embedded Software', Springer, 2010.

[Esterel2011] Esterel Technologies, Scade Language Tutorial, 2011 (available only with Scade Suite).

[Halbwachs1993] N. Halbwachs. Synchronous Programming of Reactive Systems. Springer, 1993.

[HamonRushby2007] Grégoire Hamon and John Rushby. An Operational Semantics for Stateflow. International Journal on Software Tools for Technology Transfer (STTT), 2007.

[Harel1984] D. Harel. Statecharts, a visual approach to complex systems. Dept. of Applied Math. Weizmann Institute of Science, Rehovot, Israel, 1984. Revised edition: Statecharts: a visual formalism for complex systems. Science of Computer Programming, 1987.

[JoishaBanerjee2006] P. Joisha and P. Banerjee, An algebraic array shape inference system for MATLAB, ACM ToPLaS 28(5), 2006.

[Keller&Coll2010] G. Keller, M. Chakravarty, R. Leshchinskiy, S. Peyton Jones, B. Lippmeier, Regular, Shape-polymorphic, Parallel Arrays in Haskell, ICFP, 2010, https://hackage.haskell.org/package/repa

[MaraninchiRémond1998] F. Maraninchi and Y. Rémond. Mode-automata: About modes and states for reactive systems. European Symposium on Programming, 1998.

[PothonBordin2013] F. Pothon and M. Bordin, Towards the qualification of open-source code generators: project P, int. conf. Certification Together, 2013.

[Scaife&Coll2004] N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, and F. Maraninchi. Defining and translating a "safe" subset of Simulink/Stateflow into Lustre. International Conference on Embedded Software (EMSOFT), 2004.

MBSE with the ARCADIA Method and the Capella Tool

Pascal ROQUES, PRFC

24 rue de la Digue

31170 Tournefeuille

pascal.roques@prfc.fr

Keywords:
Modeling, Model, MBSE, System Engineering, DSML, Modeling Tool, ARCADIA, Capella

Abstract:

Much more than just yet another modelling tool, Capella [1] is a model-based engineering solution that has been successfully deployed in a wide variety of industrial contexts. Based on a graphical modelling workbench, it provides systems, software and hardware architects with rich methodological guidance relying on ARCADIA, a comprehensive model-based engineering method.

The ARCADIA/Capella DSML is inspired by UML/SysML and NAF standards, and shares many concepts with these languages. It is the result of an iterative definition process driven by systems and software architects working in a broad spectrum of business domains (transportation, avionics, space, radar, etc.). It enforces an approach structured on successive engineering phases which establishes clear separation between needs (operational need analysis and system need analysis) and solutions (logical and physical architectures), in accordance with the IEEE 1220 standard.

In this paper, we will explain why a lot of industrial companies, such as Airbus, Airbus DS and Areva, are currently interested in using Capella and running modeling pilot projects with it.

Introduction

Much more than just yet another modelling tool, Capella is a model-based engineering solution that has been successfully deployed in a wide variety of industrial contexts. Based on a graphical modelling workbench, it provides systems, software and hardware architects with rich methodological guidance relying on ARCADIA, a comprehensive model-based engineering method:

- Ensure engineering-wide collaboration by sharing the same reference architecture
- Master the complexity of systems and architectures
- Define the best optimal architectures through trade-off analysis
- Master different engineering levels and traceability with automated transition and information refinement

Referring to the well-known three pillars of MBSE, we could say that ARCADIA provides both a modeling language and a modeling approach, and that Capella knows perfectly the language and the method.

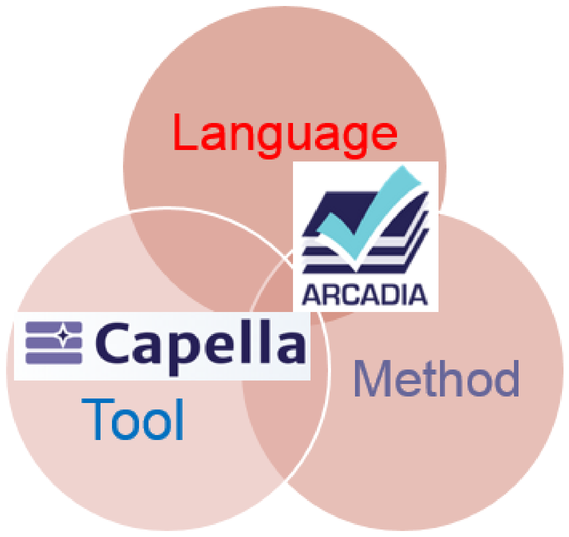


Figure 1: The three pillars of MBSE with ARCADIA/Capella

The ARCADIA Modeling Approach

ARCADIA (ARChitecture Analysis and Design Integrated Approach) is a Model-Based engineering method for systems, hardware and software architectural design. It has been developed by Thales between 2005 and 2010 [2] through an iterative process involving operational architects from all the Thales business domains (transportation, avionics, space, radar, etc.).

It enforces an approach structured on successive engineering phases which establishes clear separation between needs (operational need analysis and system need analysis) and solutions (logical and physical architectures), in accordance with the IEEE 1220 standard.

ARCADIA recommends three mandatory interrelated activities, at the same level of importance:

- Need Analysis and Modeling
- Architecture Building and Validation
- Requirements Engineering

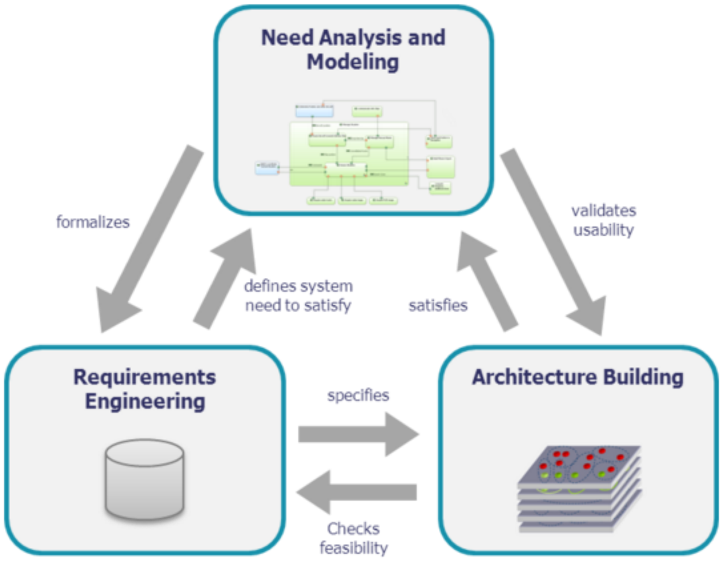


Figure 2: ARCADIA three mandatory interrelated activities

Steps and activities of the method have been defined precisely and tested on real projects inside Thales for several years. To summarize briefly, the main messages are the following ones:

- Besides requirement engineering, drive an operational need analysis, describing final user expectations, usage conditions, and realistic IVVQ conditions, and a system need analysis, describing both the requested behavior of the system under study and its external interfaces
- Structure the system and build a logical architecture, by searching for the best compromise between design drivers and non-functional constraints. Each viewpoint deals with a specific concern such as functional consistency, interfaces, performances, real time, safety, security, integration, reuse, cost, risk, schedule, and the ease of adaptation
- Secure development and IVVQ through a physical architecture which deals with technical and development issues, favoring separation of concerns, efficient and safe component interaction

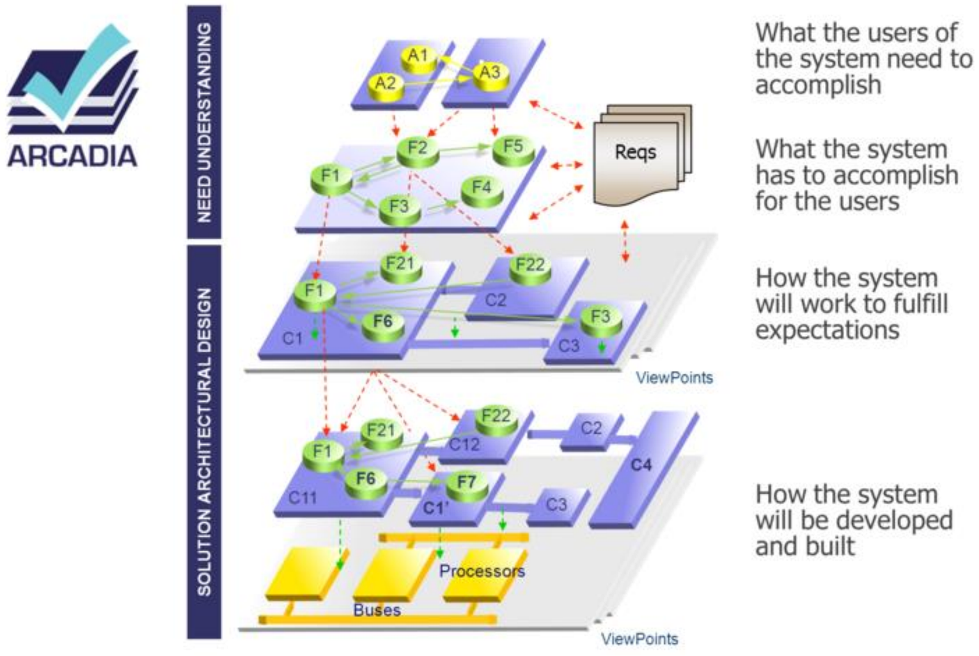

ARCADIA
NEED UNDERSTANDING
SOLUTION ARCHITECTURAL DESIGN
A1
A2
A3
F1
F2
F3
F4
F5
Reqs
F21
F22
F6
C1
C2
C3
ViewPoints
F7
C11
C12
C1'
C4
Processors
Buses
What the users of the system need to accomplish
What the system has to accomplish for the users
How the system will work to fulfill expectations
How the system will be developed and built

Figure 3: ARCADIA engineering levels

Of course, these messages are very similar to the recommendations of the INCOSE SE Handbook [3].

The ARCADIA Domain Specific Modeling Language (DSML)

The ARCADIA DSML is inspired by UML/SysML and NAF standards, and shares many concepts with these languages. But a Domain-Specific Modeling Language was preferred in order to ease appropriation by all stakeholders. ARCADIA is mostly based on functional analysis, and then allocation of the functions to components [4].

The richness of the ARCADIA DSML is comparable to SysML [5] with about ten different diagram types including data flow diagrams, scenario diagrams, states and modes diagrams, component breakdown diagrams, functional breakdown diagrams, etc. Let us give a few examples of concepts and diagrams proposed by ARCADIA at different levels on a simple case study in the meteorological domain:

Operational Analysis

A very important diagram at this level is called the Operational Architecture Blank diagram (OAB). It captures the allocation of Operational Activities to Operational Entities.

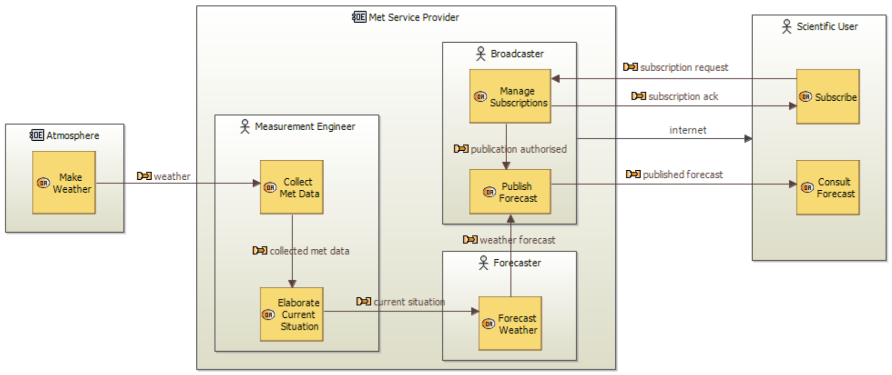


Figure 4: Operational Architecture Blank diagram (OAB) example

System Analysis

Dataflow diagrams are available in all Arcadia engineering levels. They represent information dependency between functions. These diagrams provide rich mechanisms to manage complexity: Computed simplified links between high-level Functions, categorization of Exchanges, etc. Functional Chains can be displayed as highlighted paths.

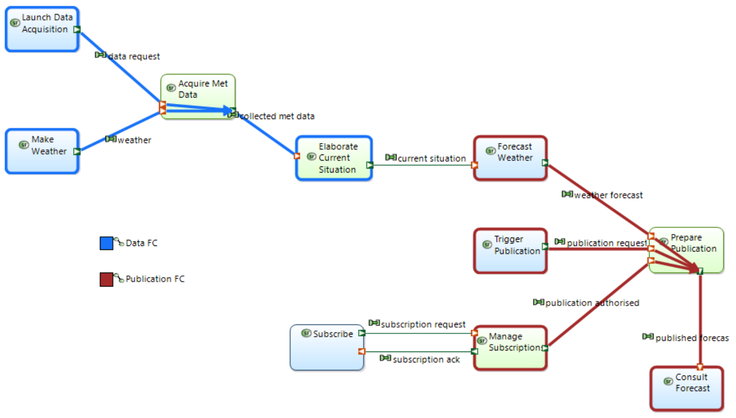


Figure 5: System Data Flow Blank diagram (SDFB) example

Architecture diagrams are used in all Arcadia engineering phases. Their main goal is to show the allocation of Functions onto Components. Functional Chains can be displayed as highlighted paths. In System Need Analysis, these diagrams contain one box representing the System under study plus the Actors.

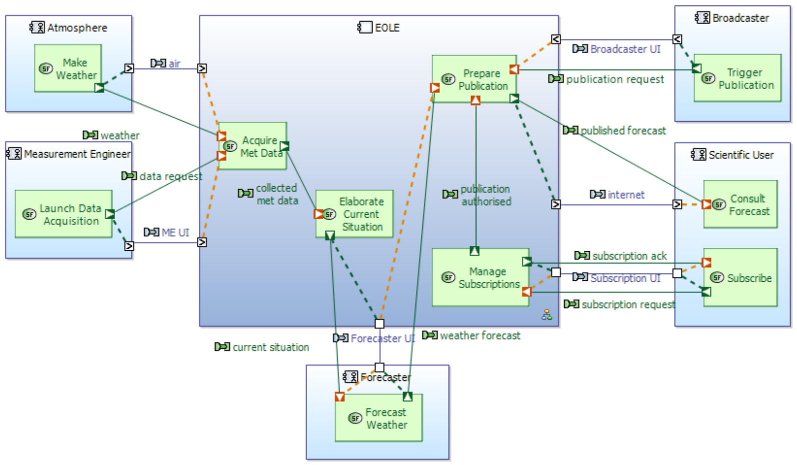


Figure 6: System Architecture Blank diagram (SAB) example

Logical Architecture

To give an example of a different diagram type, let us switch to a Scenario Diagram.

ARCADIA defines several kinds of scenario diagrams: Functional Scenarios (lifelines are Functions), Exchange Scenarios (lifelines are Components/Actors while sequence messages are Functional or Component Exchanges), Interface Scenarios (lifelines are Components/Actors while sequence messages are Exchange Items). Modes, States and Functions can also be displayed on these diagrams, which are also available at all engineering levels.

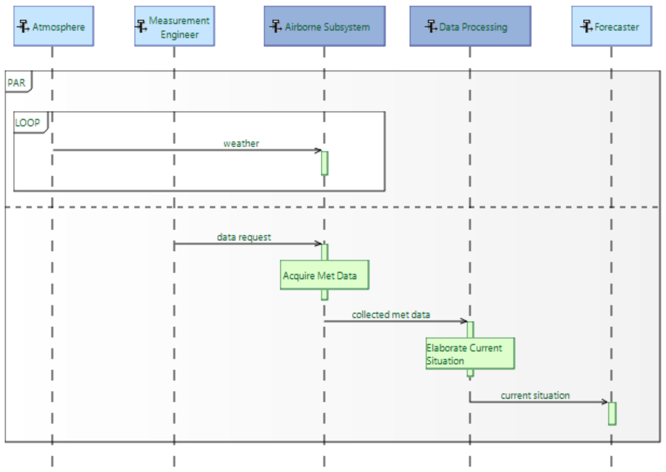


Figure 7: Logical Exchange Scenario diagram (LES) example

Physical Architecture

At this level, we also use the preceding types of diagrams, but we will show different types once again. Tree diagrams represent breakdowns of either Functions or Components.

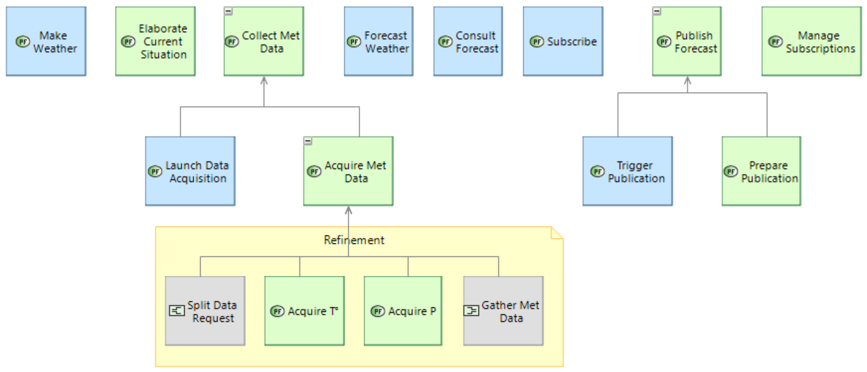
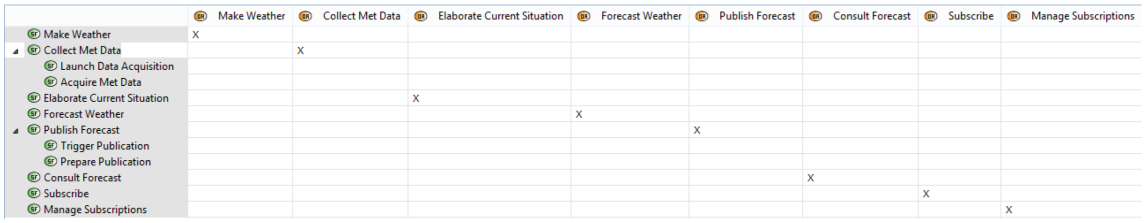


Figure 8: Physical Functional Breakdown diagram (PFBD) example

Matrix views are also available to display the different kinds of relationship between model elements. At each level, for example, two matrices are available showing the realization relationship to the upper level elements.



	Make Weather	Collect Met Data	Elaborate Current Situation	Forecast Weather	Publish Forecast	Consult Forecast	Subscribe	Manage Subscriptions
Make Weather	X							
Collect Met Data		X						
Launch Data Acquisition								
Acquire Met Data								
Elaborate Current Situation			X					
Forecast Weather				X				
Publish Forecast					X			
Trigger Publication								
Prepare Publication								
Consult Forecast						X		
Subscribe							X	
Manage Subscriptions								X

Figure 9: System Functions / Operational Activities Matrix example

To summarize, the ARCADIA DSML covers all aspects of standard architecture modelling in each engineering phase including:

- Capability-driven model organization with scenarios and functional chains
- Functional analysis and allocation to components and resources
- Interfaces, bit-precise data models, behaviors, etc.

An overview of the ARCADIA main concepts through the engineering levels is given by the next figure. Additional important concepts such as scenarios, states and modes, classes, capabilities, etc. are not shown but also available to the modeler.

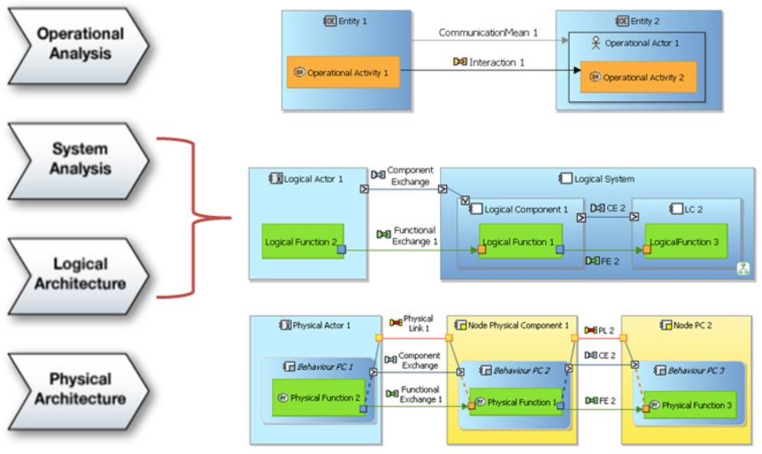
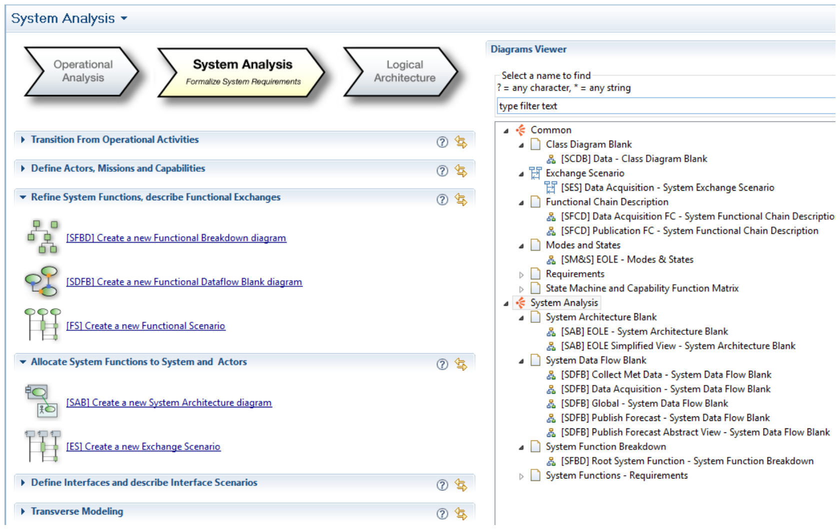


Figure 10: Summary of ARCADIA main concepts

The Capella Modeling Tool

The Capella workbench is an Eclipse application implementing the ARCADIA method providing both a Domain Specific Modeling Language (DSML) and a dedicated toolset.

A very interesting feature of Capella consists in an embedded methodology browser, reminding ARCADIA principles to the user and providing efficient methodological guidance. This activity browser provides a methodological access to all key activities of Capella, and thus to the creation of all main diagrams, level by level. It is the main entry point to a model and is both meant for beginners and power users.


System Analysis

- Transition From Operational Activities
- Define Actors, Missions and Capabilities
- Refine System Functions, describe Functional Exchanges
  - [SFBD] Create a new Functional Breakdown diagram
  - [SDFB] Create a new Functional Dataflow Blank diagram
  - [FS] Create a new Functional Scenario
- Allocate System Functions to System and Actors
  - [SAB] Create a new System Architecture diagram
  - [ES] Create a new Exchange Scenario
- Define Interfaces and describe Interface Scenarios
- Transverse Modeling

Diagrams Viewer

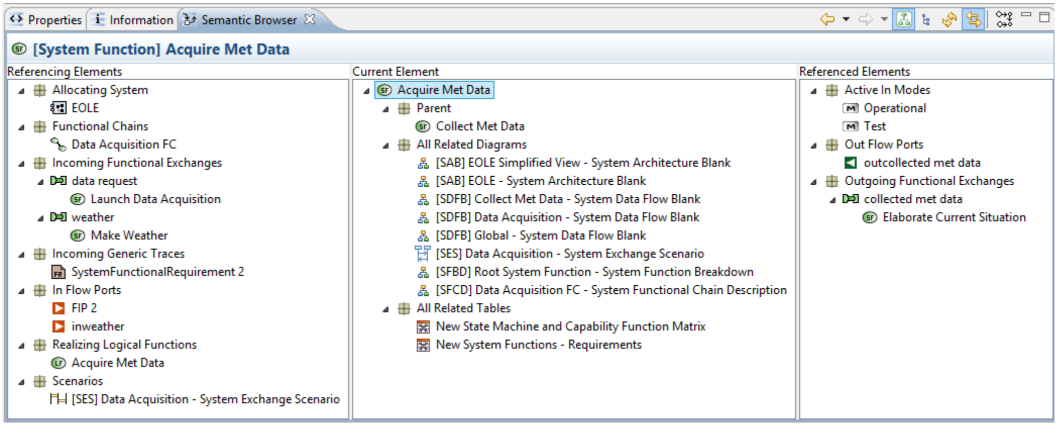
Select a name to find
? = any character, * = any string
type filter text

- Common
  - Class Diagram Blank
    - [SCDB] Data - Class Diagram Blank
  - Exchange Scenario
    - [SES] Data Acquisition - System Exchange Scenario
  - Functional Chain Description
    - [SFCD] Data Acquisition FC - System Functional Chain Descriptio
    - [SFCD] Publication FC - System Functional Chain Description
  - Modes and States
    - [SM&S] EOLE - Modes & States
  - Requirements
  - State Machine and Capability Function Matrix
- System Analysis
  - System Architecture Blank
    - [SAB] EOLE - System Architecture Blank
    - [SAB] EOLE Simplified View - System Architecture Blank
  - System Data Flow Blank
    - [SDFB] Collect Met Data - System Data Flow Blank
    - [SDFB] Data Acquisition - System Data Flow Blank
    - [SDFB] Global - System Data Flow Blank
    - [SDFB] Publish Forecast - System Data Flow Blank
    - [SDFB] Publish Forecast Abstract View - System Data Flow Blank
  - System Function Breakdown
    - [SFBD] Root System Function - System Function Breakdown
  - System Functions - Requirements

Figure 11: Capella Methodological Activity Browser

As graphical representations of elements play a key role in communication, Capella relies on a consistent color scheme. In particular, all function-related elements are green, and all component-related elements are blue. This favors enhanced model readability for all stakeholders (architects, V&V practitioners, specialty engineers, managers, etc.).

Another very useful feature of Capella is the capability to navigate inside the model elements (independently of the diagrams) through a contextual semantic browser. More practical than the standard hierarchical view of the model, the semantic browser instantaneously provides the context of model elements trough meaningful queries. It is the preferred way to navigate in models and diagrams and to quickly analyze the relationships between model elements.


Properties | Information | Semantic Browser

[System Function] Acquire Met Data

Referencing Elements
- Allocating System
  - EOLE
- Functional Chains
  - Data Acquisition FC
- Incoming Functional Exchanges
  - data request
    - Launch Data Acquisition
  - weather
    - Make Weather
- Incoming Generic Traces
  - SystemFunctionalRequirement 2
- In Flow Ports
  - FIP 2
  - inweather
- Realizing Logical Functions
  - Acquire Met Data
- Scenarios
  - [SES] Data Acquisition - System Exchange Scenario

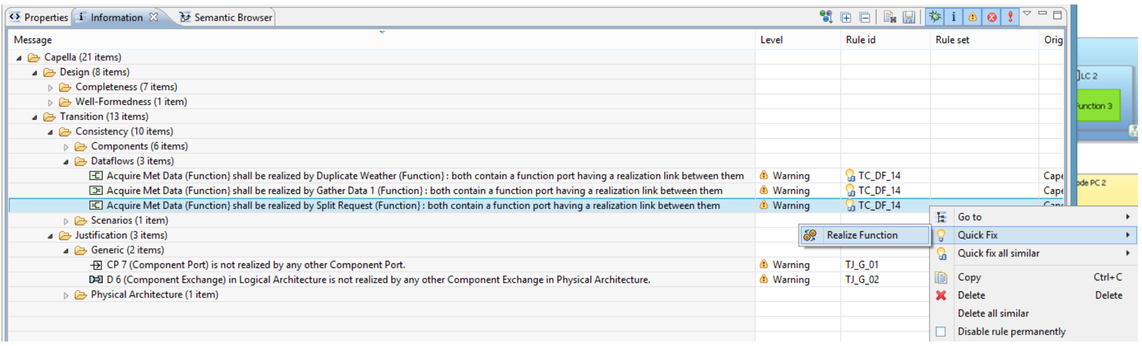
Current Element
- Acquire Met Data
  - Parent
    - Collect Met Data
  - All Related Diagrams
    - [SAB] EOLE Simplified View - System Architecture Blank
    - [SAB] EOLE - System Architecture Blank
    - [SDFB] Collect Met Data - System Data Flow Blank
    - [SDFB] Data Acquisition - System Data Flow Blank
    - [SDFB] Global - System Data Flow Blank
    - [SES] Data Acquisition - System Exchange Scenario
    - [SFBD] Root System Function - System Function Breakdown
    - [SFCD] Data Acquisition FC - System Functional Chain Description
  - All Related Tables
    - New State Machine and Capability Function Matrix
    - New System Functions - Requirements

Referenced Elements
- Active In Modes
  - Operational
  - Test
- Out Flow Ports
  - outcollected met data
- Outgoing Functional Exchanges
  - collected met data
    - Elaborate Current Situation

Figure 12: Capella Contextual Semantic Browser

Capella can go further than traditional modeling tools thanks to its knowledge of ARCADIA. For instance, the tool will check that each model element at a given engineering level is realized by a similar element at the next engineering level.

Capella organizes model checking rules in several categories: integrity, design, completeness, traceability, etc. Architects can define validation profiles focusing on different aspects. Whenever possible, quick fixes provide fact and automated solutions.


Properties | Information | Semantic Browser

Message	Level	Rule id	Rule set	Orig
Capella (21 items)				
Design (8 items)				
Completeness (7 items)				
Well-Formedness (1 item)				
Transition (13 items)				
Consistency (10 items)				
Components (6 items)				
Dataflows (3 items)				
Acquire Met Data (Function) shall be realized by Duplicate Weather (Function) : both contain a function port having a realization link between them	Warning	TC_DF_14		Cape
Acquire Met Data (Function) shall be realized by Gather Data 1 (Function) : both contain a function port having a realization link between them	Warning	TC_DF_14		Cape
Acquire Met Data (Function) shall be realized by Split Request (Function) : both contain a function port having a realization link between them	Warning	TC_DF_14		Cap
Scenarios (1 item)				
Justification (3 items)				
Generic (2 items)				
CP 7 (Component Port) is not realized by any other Component Port.	Warning	TJ_G_01		
D 6 (Component Exchange) in Logical Architecture is not realized by any other Component Exchange in Physical Architecture.	Warning	TJ_G_02		
Physical Architecture (1 item)				

Figure 13: Capella Model Checking results example

As Capella was progressively specified and enhanced by using earlier versions of the tool on Thales internal projects, it contains a lot of very efficient features such as:

- Automatic computation of graphical simplifications (for instance information exchanges between lower-level functions can be automatically displayed on higher-level functions)
- Automated contextual diagrams: content is automatically updated according to preselected model elements and predefined semantic rules
- Filters: enable the user to show simplified views of a given diagram by selecting display options and automatically hiding / showing specific model elements

Another advanced feature of Capella is the ability to create reusable model elements, either simple ones like types and classes, or complex ones, such as complete physical components with ports, functions, etc. A Replicable Elements Collection (REC, for Record) is a definition of an element which can be reused in multiple contexts / models. A Replica (RPL, for Replay) is an instantiation of a REC. RECs can be packaged in external libraries, which can be shared between several projects.

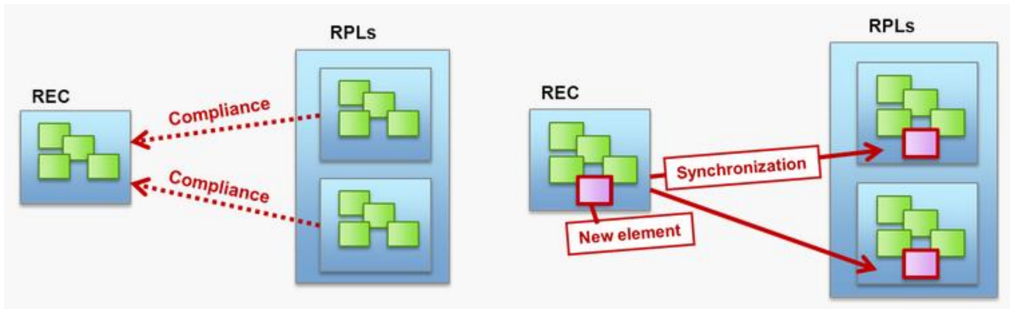


Figure 14: Capella Replicable Elements Mechanism

To widen the perspective, Capella does not work in isolation, but on the contrary fits into a wider engineering landscape, as many bridges can be developed to:

- Initialize Capella models from upstream engineering outputs (typically coming from Architecture Frameworks such as NAF)
- Confront architecture models to specialty engineering tools (performance, safety, etc.)
- Iteratively populate downstream engineering (subsystems, code generation, etc.)

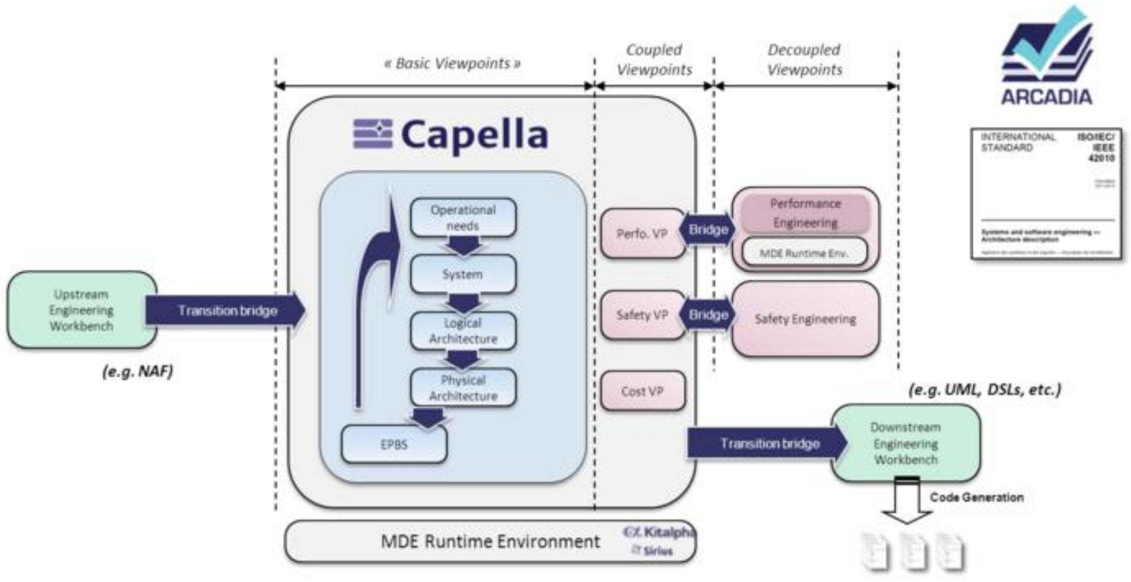


Figure 15: Capella “Big Picture”

SysML with a tool vs ARCADIA / Capella: elements of comparison

As we explained earlier, the ARCADIA DSML is inspired by UML/SysML and NAF standards, and shares many concepts with these languages. But a Domain-Specific Modeling Language was preferred in order to ease appropriation by all stakeholders, usually not familiar with general-purpose, generic languages such as UML or SysML. Previous experiments inside Thales proved that system engineers not coming from software were not at ease with the object-oriented concepts proposed by UML (and subsequently by SysML). So ARCADIA is mostly based on functional analysis, and then allocation of the functions to components. The vocabulary of the DSML has proven to be easily understood by system engineers.

So, basically, ARCADIA was defined first in Thales, from the engineering problems encountered in real projects. Then came the need for a software tool enabling to create and manage ARCADIA models. The first experiments were done using existing UML tools such as Rational Software Modeler, Objecteering and Rhapsody, and defining UML profiles on top of them [2]. At the time of these first tries, the commercial tools were not easy at all to customize, and in particular it was difficult to remove unused commands or menus. So Thales people decided to create their own tool, dedicated to ARCADIA, encouraged by the emergence of enabling technologies based on the Eclipse platform. ARCADIA definition can really be seen as the specification of the Capella modeling tool.

If we try to compare with another possible solution, namely use a standard modeling language, such as SysML, and an existing commercial tool, such as Rhapsody, we can spot several important differences. SysML and Rhapsody (as the other commercial SysML tools) are based on UML, which is a disadvantage for system engineers who have not been exposed to object-oriented concepts (notions of operation, generalization / specialization in block diagrams, and even of object flows and object nodes in the activity diagram). These object-oriented origins are clearly an obstacle to adoption by system engineers who are not familiar with the world of software development [6].

Another big problem is that SysML is only a language, and each company needs to elaborate an adapted modeling strategy. But then, how to teach the method to the modeling tool? Each commercial tool claims it offers an API to build specific add-ons, but this represents clearly a big amount of work. A prototype is provided by IBM with the Harmony for SE toolkit [7], but experiments in Thales proved that this toolkit is not more than a proof of concept and very difficult to use on real projects. For instance, the automated transitions between modeling phases were not iterative and incremental, as is the case with Capella, but merely one-shot.

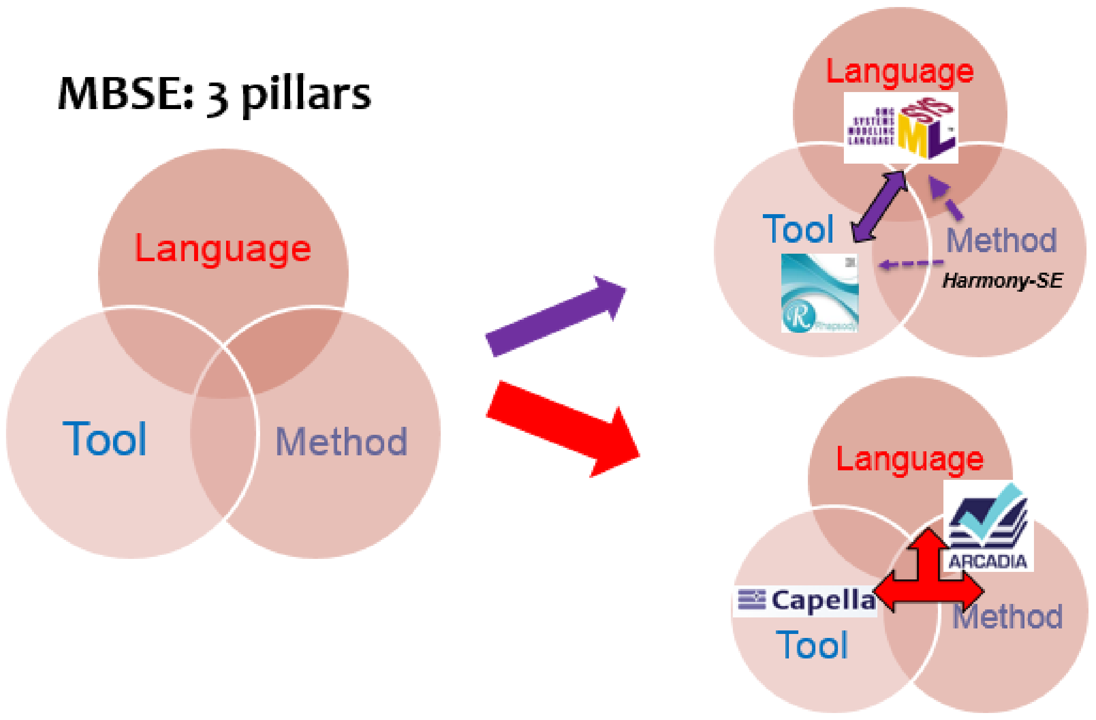


Figure 16: MBSE 3 pillars implementation: a comparison

Conclusion

The development of Capella (called Melody then) started in Thales in 2008 after a few years of experience with development of profile-based UML/SysML solutions. It is now widely deployed on operational projects in all Thales domains worldwide (defense, aerospace, space, transportation, identity and security, etc.).

Growing a community of interest and of users is a major objective of the Capella open sourcing initiative [1]. The goal is to favor the emergence of an ecosystem of organizations, including industries that would drive the Capella roadmap according to operational needs, service and technology suppliers that would develop their business around the solution, and academics that would pave the future of the engineering ecosystem.

The 3-years Clarity consortium [8] is dedicated to building this ecosystem. Since the start of the Clarity project, one year ago, such big industrial companies as Airbus, Airbus DS and Areva have already begun to experiment Capella internally. In the meantime, technology providers such as Artal, All4Tec, Scilab, have also begun to work in order to bridge Capella with simulation tools, safety engineering tools, etc. The rationale behind Clarity is that a strong adoption of this industrial solution will bring a major competitive advantage for industrial actors but also for technology and service providers.

References

[1] https://www.polarsys.org/capella/index.html

[2] V. Normand, D. Exertier, “Model-driven systems engineering: SysML & the MDSysE approach at Thales”, in “Model Driven Engineering for distributed real-time embedded systems”, John Wiley & Sons, Sept. 2005, ISBN 9781905209323

[3] INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th Edition, Wiley, 2015

[4] J.-L. Voirin, “Modelling languages for Functional Analysis put to the test of real life”, CSDM, Paris, 2012

[5] OMG, Systems Modeling Language (SysML), Version 1.4, September 2015 (http://www.omg.org/spec/SysML/1.4/)

[6] J. Aracic, P. Roques, “Select and deploy a conceptual modelling language. Some Keys.” http://blogs.crescendo-technologies.com

[7] https://www.youtube.com/watch?v=axX6wwY3puQ

[8] http://www.clarity-se.org/overview

Model Driven Engineering with Capella and AADL

Bassem Ouni[1], Pierre Gaufillet[1, a], Eric Jenn[1, b], Jérôme Hugues[2]

1: IRT Saint-Exupéry, 118 route de Narbonne, 31042 Toulouse, France

2: ISAE, 10 Avenue Edouard Belin – BP 54032 - 31055 TOULOUSE CEDEX 04

1

Abstract— **The development of real time embedded equipments is a challenging task that requires the elaboration of multiple models in several domains, notably system, electronics and software, spanning a large spectrum of multiple abstraction levels and viewpoints: structural, behavioral, dependability, etc. These models serve various purposes: specification, design, evaluation or verification and validation. Today, no single modeling language and environment covers all these aspects. While Capella – an open source modeling language and environment for system engineering developed by Thales – fits well to the most early stages of the development process, AADL – the *Architecture Analysis and Design Language* defined by the *Society of Automotive Engineers* – provides powerful capabilities to describe and analyze the design artifacts of the software point of view that appear during the latest phase of the design. This is why they have both been selected in the project INGEQUIP of IRT Saint Exupéry. While using different modeling languages for different purpose is perfectly acceptable in a development process, it is important to guarantee that information remain consistent across all models. This is why building a formalized bridge between Capella and AADL is an essential piece of INGEQUIP process. In this paper, after an introduction to the context of INGEQUIP, the high level semantics of Capella and AADL are compared. The mapping used in INGEQUIP between Capella physical models and AADL abstract models is then described. The whole approach is illustrated by some elements coming from the design of TwIRTee – the robotic demonstrator of INGEQUIP – before concluding.**

Keywords—Real time, Embedded systems, Model transformations, Performance evaluation, Constraints verification, Capella, AADL.

I. INTRODUCTION

This work is achieved under the INGEQUIP project at the Toulouse *Institut de Recherche Technologique* (IRT) Saint-Exupéry. IRTs are new research structures established under the auspices of the French *Agence Nationale de la Recherche* (ANR). IRTs are aimed at favouring the transfer of innovation from laboratories to industries. Towards this goal, IRTs gather engineers coming from small to large companies from various industrial domains, and researchers from public universities and national research agencies. As an example, INGEQUIP covers the space, aeronautics and automotive systems domains.

The goal of INGEQUIP is to study and propose solutions for supporting closely integrated development of the main technical domains involved in embedded equipment engineering – system, eleectronics and software engineering. A key element for reaching such goal is to ensure the continuity and consistency of information in the whole chain of activities. In INGEQUIP, the choice has been to obtain this property by relying on models and model transformations. The set of requirements regarding an equipment is usually divided into two categories: functional requirements and non functional requirements. Functional requirements include the system's behavior, capabilities and characteristics as specified by stakeholders whereas non-functional properties or requirements define criteria that can be used to evaluate the operations of the system. In order to design the system and associate to the design elements the realized functional and non functional requirements, several modeling languages are available among which AADL

a Seconded from Airbus, 316 route de Bayonne, 31000 Toulouse, France.
b Seconded from Thales Avionics, 105 avenue du Général Eisenhower, 31100 Toulouse, France.

[1] – *Architecture and Analysis Design Language*, AUTOSAR [2], Capella [3], EAST-ADL [4], SysM [5] and UML [6] have been considered. While Capella, EAST-ADL and SysML fit system engineering, AADL, AUTOSAR and UML are focused on software engineering. Finally the couple Capella/AADL has been chosen in INGEQUIP because they provide most viewpoints commonly used by designers of embedded equipment at system and software levels: functional breakdown, logical and physical architecture, software dynamic and static architecture, formal behavioral descriptions; they are supported by a number of tools widely available: Capella and OSATE for edition, formal behavioral analysis tools like FIACRE and Tina [7], Cheddar [8] and code generations like OCARINA [9]). They are also based on open technologies like the Eclipse Modeling Framework which allows to develop easily tools extensions.
Consistently with the orientation of INGEQUIP, a transformation has then been defined and developed for ensuring a seamless transition from system engineering stage to software engineering stage.

In this paper, we therefore begin in section 2 by introducing the state of the art of engineering models transformation. Then, section 3 presents a first comparison between the semantics of Capella and AADL. The mapping of the transformation between Capella physical model and AADL abstract model is then described in section 4. The whole approach is illustrated by some elements coming from the design of TwIRTee – the robotic demonstrator of INGEQUIP – in section 5 before concluding in section 6.

II. RELATED WORKS

In order to achieve an early analysis of the specification, the verification of functional and non-functional properties of the system, and even code generation for the targeted hardware platform, several studies have proposed comparable transformations to AADL models.
In [10], Brun et al. introduce an approach for translating UML/MARTE detailed design into AADL design. The proposed work focuses on the transformation of the thread execution and communication semantics and does not cover transformation of embedded system components, such as equipment parts.
Turki et al. [11] propose a methodology for mapping MARTE model elements to AADL components. They focus on the issues related to modelling architectures. This transformation flow does not consider issues related to the mapping of MARTE properties to AADL property. The syntactic differences between MARTE and AADL are well handled by the transformation rules provided by ATL tool.
In [12], AADL is used for modelling the properties of embedded system architecture, including the application's tasks, the hardware platform and the operating system services in order to characterize the energy overhead of embedded operating system . The authors propose an AADL model transformation in order to be exploited by a multiprocessor simulation tool named STORM (Simulation TOol for Real-time Multiprocessor scheduling). AADL provides hardware and software architectures together with the scheduling policy; STORM simulates the system behaviour using all the characteristics (task execution time, processor functioning conditions, etc.) in order to obtain the chronological track of all the scheduling events that occurred at run time, and compute various real-time metrics in order to analyse the system behaviour and performances from various point of views. Then, this work has been extended in [13] and a system design exploration methodology has been proposed to verify system requirements when allocating applicative tasks to the processors using a set of tools: RDALTE for the definition and analysis of system requirements and QAML for quantitative analysis.
In this paper, the proposed approach for transformation of Capella to AADL models target to cover the various levels of abstraction when modeling systems. We take into account the system behavior and the hardware/software mapping. Next section will detail the transformation flow and how it exploits the complementarity of Capella and AADL in order to cover various embedded system aspects at high level modeling step.

III. PROPOSED METHOD

As Capella and AADL partially overlap, as shown in the diagram (Figure 1) below, the first question that has to be answered before defining the transformations from Capella to AADL is the level at which this transformation should be performed. Capella is clearly positioned on the most abstract part of the system development process with the *Operational Analysis*, focusing on the capture of stakeholders' needs, and the *System analysis*, focusing on the functional definition of the system. As for AADL, it doesn't offer support for such kind of analysis. As they deal with the system's architecture, Capella's logical and physical models share lots of concepts with AADL, in particular the capability to express structural refinement. AADL however goes further by providing not only system-oriented abstract components, but also explicit hardware components – *device*, *processor*, *memory*, *bus* – and explicit software components – *process*, *thread*, *subprogram*.

Moreover, AADL proposes a formal models of communication and execution semantics that is a prerequisite for any low-level behavioral analysis (e.g., schedulability analysis).

Capella provides system modeling capabilities at several layers of abstraction:

- At operational level, the customer needs, the actors, the missions and the activities are described.
- At system level, a Capella model defines how system can satisfy the former operational need.
- Capella logical level modeling starts from functional and non-functional analysis and builds one or several decompositions of the system into logical components.
- The building of logical components is performed at physical level: the "final" architecture of the system introducing architectural patterns, services and components, and it makes the logical architecture evolve according to implementation, constraints and choices.

Figure 2 summarizes the relationship between functional, logical and physical architecture in a Capella model. In this work, we interest to parse the physical architecture of the Capella model as it includes functional, logical and physical components.

Whatever the abstraction level of the considered components is, AADL also brings the capability to specify additional information compared to Capella, like functional or non-functional properties. Consequently, the simplest articulation between the two languages is at the levels of the logical or physical architectures. Considering that Capella is well adapted to describing the logical architecture and a first abstract level of physical architecture, and that it is a good practice to limit the risk of data duplication and inconsistency between Capella and AADL models, the best solution is to delay as far as possible the transfer of information. As shown in figure 3, the model transformation will therefore take as input the most detailed physical architecture in Capella, and the design process will go on in AADL.

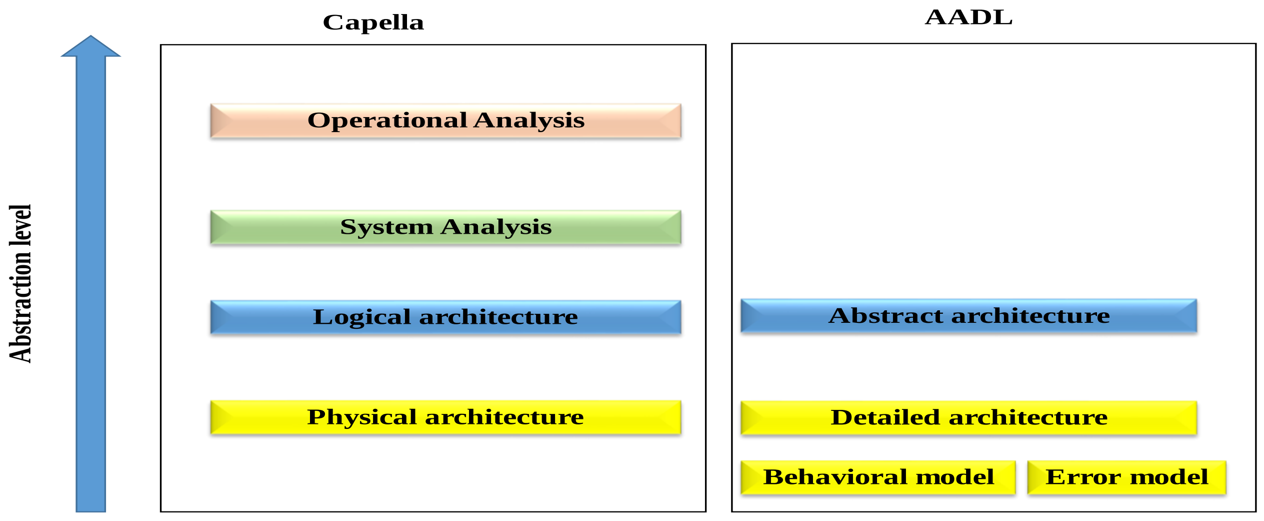


Figure 1- Capella and AADL positioning

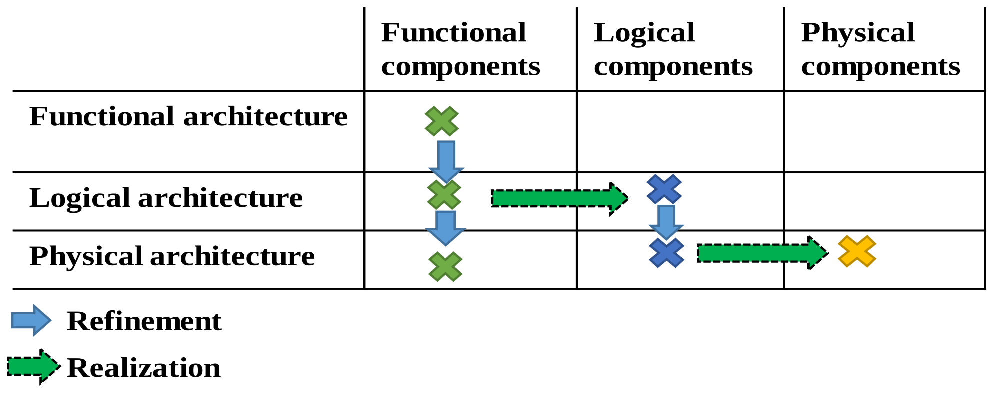


Figure 2- Capella functional, logical and physical architectures

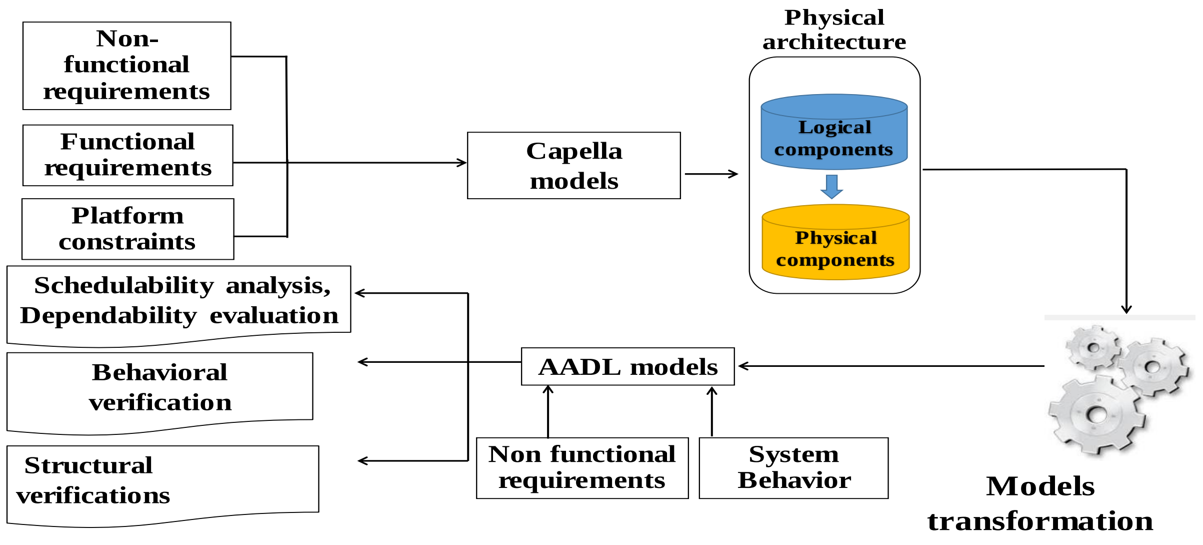


Figure 3 - Capella/AADL data flow

IV. THE CAPELLA TO AADL TRANSFORMATION

A. *Transformation definition*

To transform Capella to AADL models, we explore the contents of a Capella model and generate the appropriate AADL code following the Capella/AADL concepts analogy proposed in Table 1. The equivalent of an AADL processor, device are Capella node physical components with respectively a software execution unit and hardware kinds. A physical component with behavior nature and software application kind is considered as process in AADL.

Physical buses components are not modeled in Capella meta-models. For this reason, using Capella ecore meta-models, we generate Capella EMF [14] code and exploit it using Java to extract physical buses from Capella model and map them to AADL buses. The determination of buses is elaborated by exploring the Capella physical components, the physical links which are the communication/transportation means linking node Physical components, and physical ports. As depicted in figure 4, the physical links connected by physical ports are gathered in “*AllconnectedLinks”* list. For each element of the list, a physical bus component, with a bus port, will be generated to bind the physical links. Then, as showed in Figure 5, buses are connected to external ports (*tip ports*) which are ports of physical components having no subcomponents. This established link is called *BusLink*.

*BusLink*s are divided into segments that don't cross physical component boundaries. These segments represent synthetic links. Synthetic ports includes the *tipports* and new ports which are generated at each intersection between *BusLinks* and physical components. The algorithm describing buses extraction is detailed in algorithm 1. The output of this algorithm is the list of synthetic ports and links that will be mapped to AADL elements.

In Capella, logical components are not stored in the physical components they are deployed on, but, in the physical system root, , For rebuilding the AADL connection path, and as depicted in figure, a virtual physical component is created instead of each logical component, a connection is established between these virtual components, then as for the buses, we split this connection into parts and we generate new Capella ports with a new property: the direction.
To transform Capella to AADL models, Accelo plugin [15] is used, it allows the generation of AADL code from input Capella graphic models. We exploit this plugin to explore the contents of a Capella model and generate the appropriate AADL code following the Capella/AADL concepts analogy proposed in Table 1.

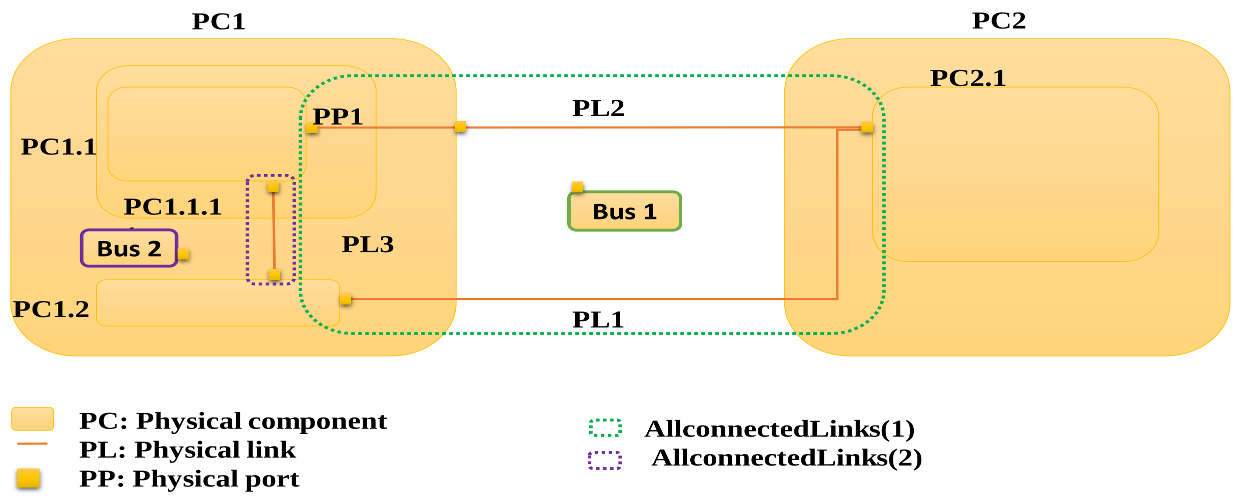


Figure 4 – Determination of connected physical links

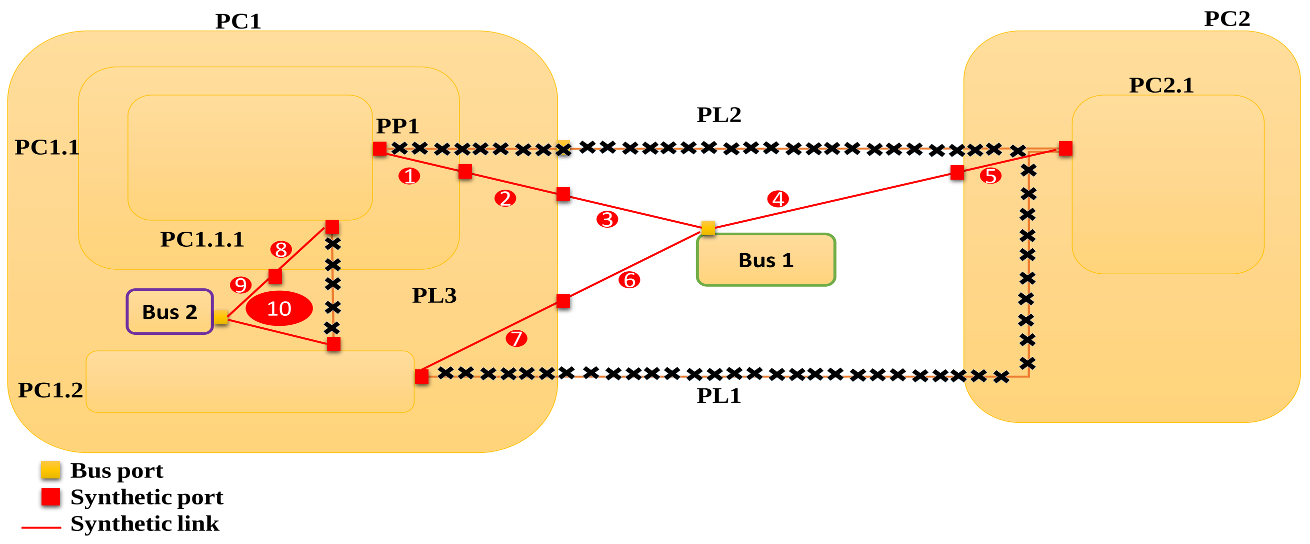


Figure 5 – Generation of synthetic links and ports

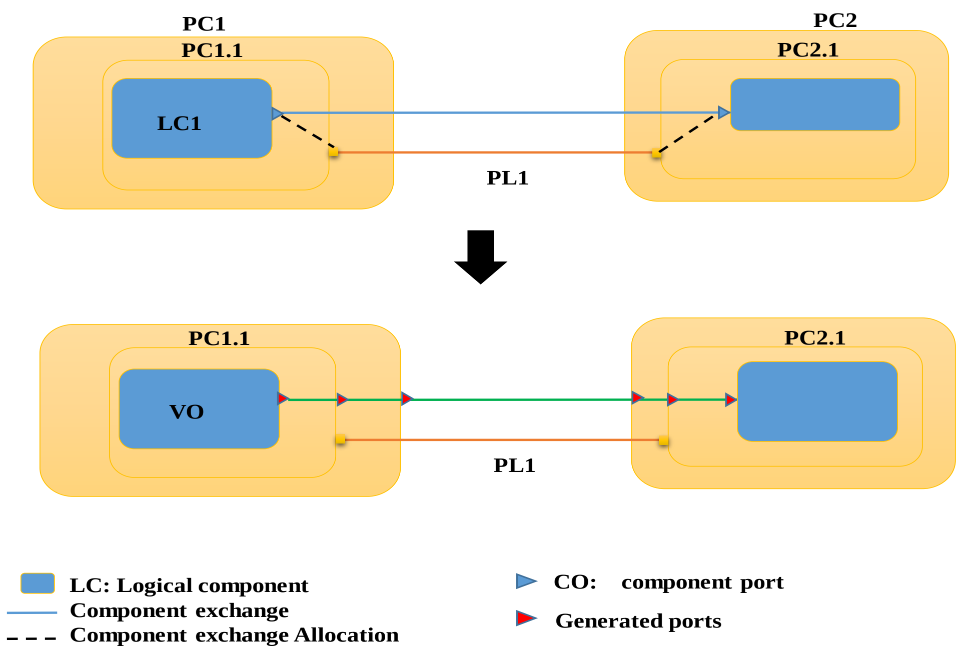


Figure 6 – Capella component exchanges transformation

Algorithm 1 – Capella Bus generation algorithm

```
1. Get the physical architecture from Project in Capella model.
2. Extract the list of physical ports pps physical links pls and a cross reference between them.
3. Determinate the list allConnectedLinks including the sets of pls connected by pps.
4. for each element of allConnectedLinks
     a. Get the category of the link
     b. Create the list of external ports tipports
     c. Create a bus instance having the same name of category
     d. Associate the bus instances with tipports in a BiMap structure bus2TipPorts (key=bus instance, value=tipports).
5. end for
6. Initialize the list of synthetic ports with the tip ports and synthetic links.
7. for each element of bus2TipPorts
     a. Search the closest common physical component ancestor PcCommonAncestor of tipPorts
     b. Create the Physical component that will model physically the bus instance: PCBus
     c. Add new properties to PCBus: isBus and Bustype property
     d. Add a port BusPort to the PCBus
     e. Add PCbus to PcCommonAncestor
     f. for each port p of bus2TipPorts
          i. Create a link from tip port p to BusPort
         ii. Divide this link into segments that don't cross Physical Component boundaries
        iii. Update the list of synthetic ports and links
     g. end for
8. end for
```

Case	Capella	Condition	AADL
A	PhysicalComponent	nature =PhysicaComponentNature::NODE and kind=PhysicalComponentKind::SOFTWARE_EXECUTION_UNIT	Processor
B	PhysicalComponent	nature =PhysicaComponentNature::NODE and kind=PhysicalComponentKind::HARDWARE	Device
C	PhysicalComponent	nature =PhysicaComponentNature::BEHAVIOR and kind=PhysicalComponentKind::SOFTWARE_APPLICATION	Process
D	PhysicalComponent	other than	System
E	PhysicalPort	See Bus extraction algorithm	Requires bus access
F	PhysicalLink		Bus access connection + Bus
G	ComponentPort	kind=ComponentPortKind::FLOW and direction=OrientationPortKind::[IN|OUT|INOUT] See Feature connection extraction algorithm	Feature
H	ComponentExchange	See Feature connection extraction algorithm	Feature connection
I	ComponentPortAllocation		Actual_connection binding_property on bus
J	Other metaclasses		N/A

Table 1 – Mapping of Capella – AADL concepts

V. APPLICATION TO THE TWIRTEE ROVER

TwIRTee is a three-wheeled autonomous rover developed within the INGEQUIP project. Its operational role is very simple: move itself on some predefined tracks from a point A to a point B (a "mission") while avoiding other rovers. To achieve this mission, it is fitted with several sensors (camera, odometry sensors, global positioning,…) and two main actuators (motors).

The development of the rover is not an objective *per se.* Indeed, TwIRTee is designed so as to cover the major topics addressed in the project namely: early validation, architecture exploration, performance prediction, and formal verification. Furthermore, it is aimed at covering issues, and functional and architectural elements specific to the three industrial domains. Accordingly, missions, functions and the architectural elements are determined so as to tackle or exercise one or several issues: for instance, “the localization” function relies partially on imaging so as to exercise hardware / software space exploration and co-design; the highly redundant architecture provides the experimental setup to perform early performance evaluations (including dependability), etc.

The computing platform of TwIRTee is composed of 2 COM/MON channels that host the main “mission” functions and one channel dedicated to power supply generation and motor control. In order to cover issues related to the development of safety-critical Man Machine Interfaces, the system also contains a remote operator station. Figure 3 and Figure 4 show respectively elements of TwIRTee design in Capella and their corresponding elements in AADL.

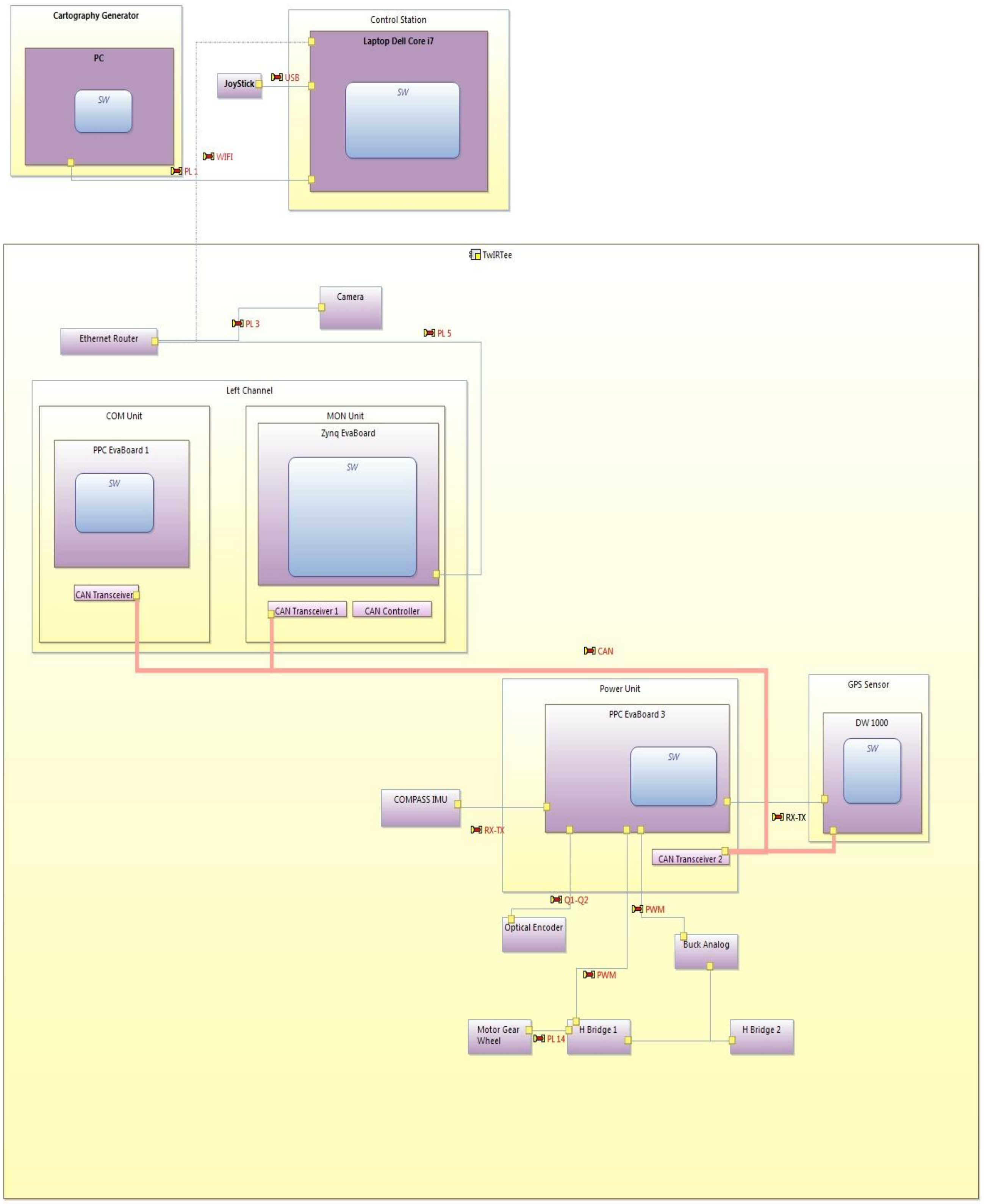


Figure 7 - Elements of TwIRTee design in Capella

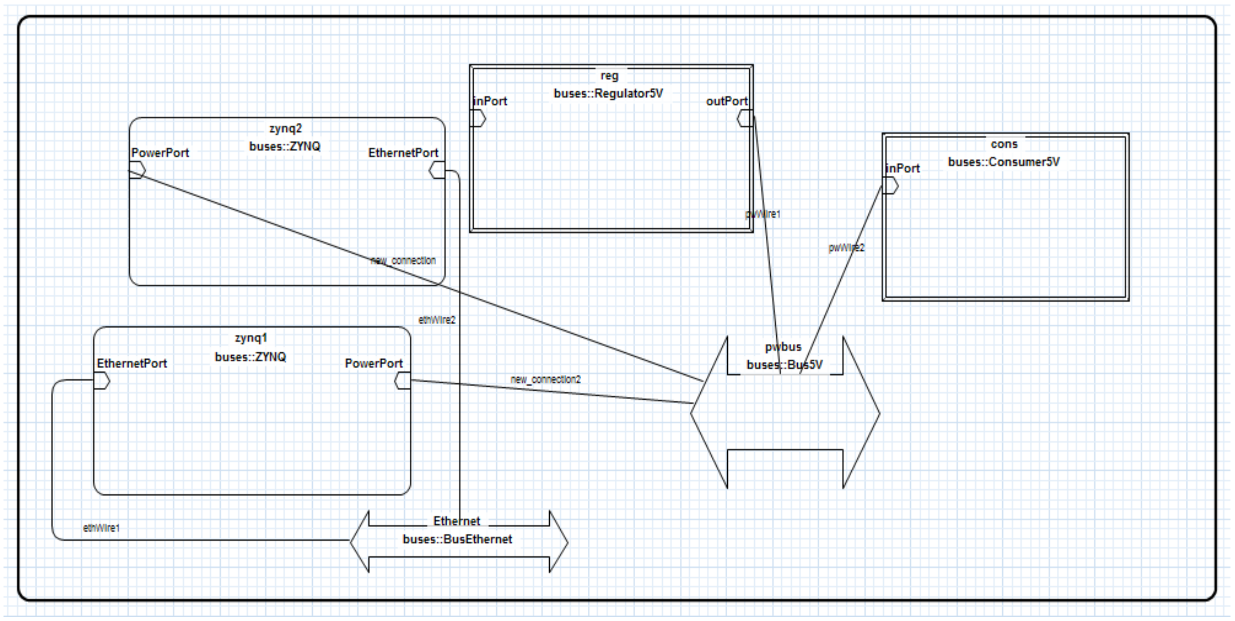


Figure 8 - Corresponding TwIRTee elements in AADL

VI. CONCLUSION

In this paper, we introduced a transformation from Capella physical architecture to a preliminary software architecture in AADL. The goal of this approach is to implement a seamless development process from system definition and design to software design and implementation. The approach has been applied and validated during the design of the TwIRTee rover demonstrator. The resulting physical model has been used to verify structural, performance (power and energy consumption), timeliness and dependability properties.

ACKNOWLEDGEMENT

The authors thank all people and industrial partners involved in the Ingequip project. This work is supported by the French Research Agency (ANR) and by the industrial partners of IRT Saint-Exupéry Scientific Cooperation Foundation (FCS): Actia Automotive, Airbus (a), Airbus Defense and Space, Continental Automotive, SAGEM, Systerel, Thales Avionics (b), ASTC Design Partners, Space Codesign Systems.

VII. REFERENCES

[1] Architecture Analysis and Design Langage (AADL), SAE standards, "Architecture Analysis and Design Langage (AADL), SAE standards," [Online]. Available: http://standards.sae.org/as5506/.

[2] "AUTOSAR," [Online]. Available: http://www.autosar.org/.

[3] "Capella tool environement and Arcadia methodology," Thales group, 2015. [Online]. Available: https://www.polarsys.org/capella/arcadia.html.

[4] "EAST-ADL," [Online]. Available: http://www.east-adl.info/Specification.html.

[5] "SysML," [Online]. Available: http://www.omgsysml.org/.

[6] "UML," [Online]. Available: http://www.uml.org/.

[7] B. Berthomieu, J.-P. Bodeveix, S. Dal Zilio, P. Dissaux, M. Filali, P. Gaufillet, S. Heim and F. Vernadat, "Formal Verification of AADL models with Fiacre and Tina," *Embedded Real-Time Software and Systems*, 2010.

[8] "Cheddar," [Online]. Available: http://beru.univ-brest.fr/~singhoff/cheddar/.

[9] Ocarina, "Ocarina," [Online]. Available: http://www.openaadl.org/ocarina.html.

[10] M. Brun, M. Faugère, J. Delatour and T. Vergnaud, "From UML to AADL: an Explicit Execution Semantics Modelling with MARTE," in *EMBEDDED REAL TIME SOFTWARE AND SYSTEMS*, Toulouse, January 2008.

[11] S. TURKI, E. SENN and D. BLOUIN, "Mapping the MARTE UML Profile to AADL," in *Models ACES-MB Workshop Proceedings*, Oslo, Norway, October 2010.

[12] B. Ouni, C. Belleudy and E. Senn, "Accurate energy characterization of OS services in embedded systems," *EURASIP Journal on Embedded Systems,* vol. 6, 2012.

[13] B. Ouni, "Thesis dissertation: High-level energy characterization, modeling and estimation for OS-based platforms," Sophia Antipolis, France, July 2013.

[14] "Eclipse Modeling Framework," [Online]. Available: https://eclipse.org/modeling/emf/.

[15] "Acceleo Project," Obeo company, 2006. [Online]. Available: http://www.eclipse.org/acceleo/.

Session 27

Simulation

Friday 29th, 09:00 – Ariane 1

Virtual Yet Precise Prototyping: An Automotive Case Study

Daniela Genius, Sorbonne Universités, UPMC Paris 06, LIP6, CNRS UMR 7606,
daniela.genius@lip6.fr

Ludovic Apvrille, LTCI, CNRS, Telecom ParisTech, Université Paris-Saclay, IMT,
ludovic.apvrille@telecom-paristech.fr

Abstract

The paper overviews a joint framework for validating and exploring complex embedded systems. The framework indeed combines AVATAR and SocLib. AVATAR is a Model Driven Engineering approach relying on SysML, and SoCLib is a virtual prototyping platform.

The main contribution lies in the new possibility to map AVATAR SysML blocks onto hardware nodes, within a newly conceived Deployment Diagram, and then to transform the latter into SoCLib models. At the AVATAR level, diagrams can be formally verified and simulated in a functional way only, that is, without considering any underlying hardware execution environment. On the contrary, the SocLib models can be simulated taking into account hardware components in an explicit way with a cycle-accurate bit accurate approach.

An automotive system is used to present the two abstraction levels, and their support by the TTool toolkit.

1. Introduction

The complexity of recent systems pushes current design techniques to their limits. While model-oriented design of complex embedded systems is nowadays a current practice in software development for embedded systems, the hardware aspects of such systems are less frequently designed using this kind of approach. Hardware is described on several abstraction levels, that are rarely represented with graphical modeling formalisms: TLM-DT (Transaction level with distributed time), CABA (Cycle/Bit Accurate), and RTL (Register Transfer Level).

The models of software components of complex embedded systems are generally tested/executed on the local host, and integrated on the target once the latter is available. Even if several virtual prototyping platforms are available for to evaluate the software in a quite realistic way before the target is available purpose, they require to re-model software elements in a different input format from the one used in models or programming languages.

AVATAR is a SysML-based environment to model the software components of complex embedded systems [1]. It is particularly adapted to these systems because (i) it proposes new temporal operators to better describe the temporal constraints of these systems and (ii) its models are formally defined, that is, formal simulations and verifications can be performed from the models. From AVATAR models, it is also possible to generate executable C/POSIX code that can be executed either on the local development platform, or on a target. In a former contribution, we already presented how that code can also be executed in the SoCLib virtual prototyping environment [2]. Yet, the joint use of both AVATAR and SocLib was not well integrated since the description of the hardware execution environment could be done only in a textual form, that is, totally outside of the SysML models.

Thus, we now offer a fully integrated way to model critical software components, to model the candidate support hardware architectures, and then to evaluate the execution of the former onto the latter using automated model-to-soclib transformation techniques. The automation of this process in a toolkit (TTool) paves the way for a easy-to-used model-based approach for design space exploration at a low-level of abstraction (e.g., CABA simulation). Figure 1 highlights the novel contributions among the existing AVATAR framework. These new contributions are displayed in red and within a red dotted line.

Section 2 presents the related work. Section 3 describes our methodology. Section 4 presents the automotive case study. Section 5 explains the details of our approach, while Section 6 discusses its limitations. The conclusions and perspective on future work is finally presented in Section 7.

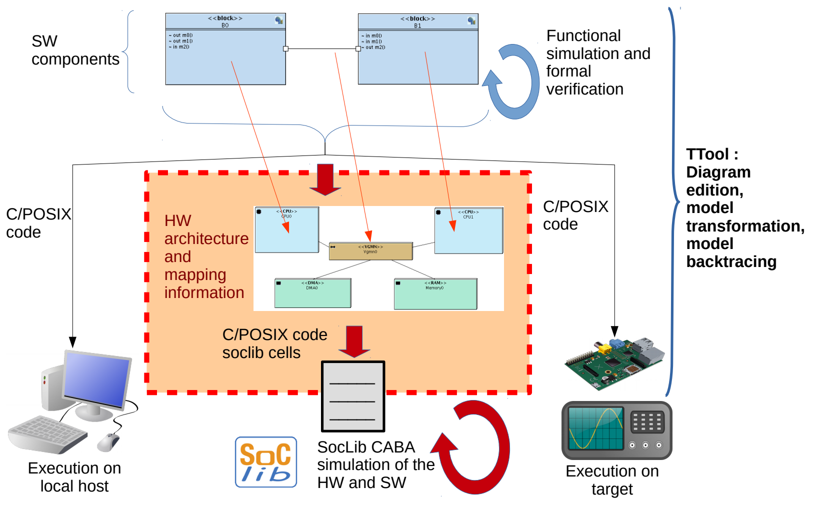

SW components
Functional simulation and formal verification
TTool : Diagram edition, model transformation, model backtracing
C/POSIX code
HW architecture and mapping information
C/POSIX code
C/POSIX code soclib cells
SocLib CABA simulation of the HW and SW
Execution on local host
Execution on target

Figure 1. Overview of model transformations for simulation, verification and code generation/execution. Bracktracing to models is not presented, but is effectively implemented in TTTool.

2. Related Work

There are several modeling environments targeting the evaluation/prototyping of functions mapped on complex hardware architectures.

Ptolemy [3] proposes a modeling environment for the integration of diverse execution models, in particular hardware and software components. If design space exploration can be performed with Ptolemy, its first intent is the simulation of the modeled systems.

In Polis [4], applications are described as a network of state machines. Each element of the network can be mapped on a hardware or a software node. This approach is more oriented towards application modeling, even if hardware components are closely associated to the mapping process. Metropolis [5] is an extension of Polis. It targets heterogeneous systems and offers various execution models. Architectural and application constraints are closely interwoven. Metropolis is based on a meta-model of a *network of concurrent objects*, with a formal semantics. Applications are described in detail and simulated with the help of instruction set simulators (ISS).

In SPADE [6], applications are modeled as Kahn processes [7], then mapped to hardware architectural models before being precisely simulated. SPADE is essentially based on RISC processor models.

Sesame [8] proposes modeling and simulation features at several abstraction levels. Preexisting virtual components are combined to form a complex hardware architecture. In contrast to Metropolis, application and architecture are clearly separated in the modeling process. Models' Semantics vary according to the levels of abstraction, ranging from Kahn process networks (KPN) for application modeling, to data flow for model refinement, and to discrete events for simulation purpose. Currently, Sesame is limited to the allocation of processing resources to application processes. It neither models memory mapping nor the choice of the communication architecture.

The ARTEMIS [9] project originates from heterogeneous platforms in the context of Philips research It addresses multimedia applications in particular, thus justifying the acronym (ARchitecTurEs and Methods for embedded MedIa Systems). It is strongly based on the Y-chart approach which has a long tradition at Philips. Simulation is done with SPADE, Sesame or a proprietary Philips tool named TSS (Tool for System Simulation). Application and architecture are clearly separated: the application produces an event trace in a file at simulation time, which is then read in by the architecture model. However, behaviors depending on timers and interrupts cannot be taken into account.

MARTE [10] shares many commonalities with our approach, in terms of the capacity to separately model communications from the pair application-architecture. For such a purpose, MARTE proposes Behavior Scenarios- and Steps (Communication Steps). However, these assets are designed for performance and timing analysis, rather than DSE. Consequently, they intrinsically lack a separation between control aspects and message exchanges as we proposed in Activity and Sequence Diagrams. Even if the UML profile for MARTE adds capabilities to model Real Time and Embedded Systems, it is not specifically targeting architectural exploration: it does not offer any methodology for that purpose, nor selected models, nor model transformation for simulation or formal verification. On the contrary, this is one important goal of our approach.

Other works based on UML/MARTE such as GASPARD [11] are dedicated to both hardware and software synthesis relying on a refinement process based on user interaction to progressively lower the level of abstraction of input models. However, such a refinement does not completely separate the application (software synthesis) or architecture (hardware synthesis) models from communications.

MDGen from Sodius [12] starts from Rhapsody, which can automatically generate software, but not hardware descriptions from SysML. SysML in Rhapsody is untimed and sequential. Also, timing and hardware specific artifacts such as clock/reset lines are generated automatically. Yet, this approach is probably closest to our present contribution, apart from the lack of hardware description.

The Architecture Analysis & Design Language (AADL [13]) is a standard from the International Society of Automotive engineer (SAE). It allows the use of formal methods for safety-critical real-time systems in avionics, automotive among other domains. It comprises a textual and a graphical representation. It does not a priori contain tool support for code generation. The architecture is modeled in a similar way as we do, e.g., the description of hardware components whose interaction is modeled by connections. Similarly to our environment, a processor model can have different underlying implementations and its characteristics can easily be changed at modeling stage. In the case of our contribution, four components are defined: processors, memories, devices, and buses. The Deployment Diagrams we present in this paper proposes a much larger variety of hardware components. Moreover, the SoClib library, already very rich in detailed models, can easily be enriched by additional components, e.g. with specific coprocessors or other interconnects than a bus, allowing for very detailed simulation with the desired degree of specificity. It does not a priori contain tool support for code generation. An approach that generates a system implementation from models using Simulink has recently been presented [14].

Capella [15] is relying on Arcadia, a comprehensive model-based engineering method. Originating from Thales and widespread in the domains of defense, space and transportation within the company, it provides architecture diagrams allocating functions to components, allocation of Behavioral Components onto Implementation Components (typically hardware, but not necessarily). The basic idea is to check the feasibility of customer requirements, called *needs*, for very large systems. On the contrary, TTool/Soclib is oriented towards co-design for a given case, like the one deonstrated in this paper. As in AVATAR, Capella also provides sequence diagrams and state machines. Capella also provides advanced mechanisms to model bit-precise data structures and relate them to Functional Exchanges, Component or Function Ports, Interfaces, etc. In this sense, it goes further than AVATAR which does not model data structures in such a precise way.

3. Methodology

The methodology of our approach is now better explained (see Figure 1). It can be seen as en extended version with regards to the one presented in [2]. New stages are denoted with a starting "*".

1. **Requirements**. Both safety and security requirements of the system are first captured with SysML Requirements Diagrams.

2. **Design**. The general structure of the software components is modeled with SysML block Diagrams. The behavior of each block is described by a state machine. That behavior can range from a quite abstract, to a more precise one manipulating e.g. data types.

3. **Functional simulation and formal verification**. The press-button approach of TTool makes it possible to perform simulations with model animation. Safety and security formal proofs can also be performed directly from the design models without prior knowledge about underlying formal verification techniques. Safety and security proofs rely on UPPAAL [16] and on ProVerif [17], respectively.

4. **Software code generation**. TTool can generate C/POSIX code from design models. The code can then be compiled and executed either for the localhost or for a given target. This code generation assumes a simple hardware system (One CPU, one memory, etc.).

5. ***Hardware architecture description and mapping**. A deployment diagram can be used to define and interconnect hardware nodes, e.g., processors, memories, buses/interconnects, I/O, interrupts and timers. The execution part of software components can be mapped onto processors. The data part of software components can be mapped onto memory banks.

6. ***Software and hardware code generation**. TTool can now generate the virtual prototyping

code from design and deployment models. The code contains both a SoCLib hardware description of the mapped platform and the software code (C/-POSIX) to be executed on that platform.

7. ***Prototyping with SoCLib**. The SoCLib simulator can be started and the code - generated and compiled during previous step - is loaded and executed like on real hardware. Debugging can be performed at two levels using both the GNU debugger, simulation traces directly displayed by TTool.

Simulation and formal proofs are meant to be executed during the first iterations on the system model. On the contrary, the prototyping of the system is expected to be performed during the last iterations, that is, on more refined models. In all cases (simulation, verification and prototyping), results are directly displayed by TTool in a SysML fashion, therefore facilitating the identification of problems directly on SysML models.

Our environment is built upon two existing modeling and simulation environments: AVATAR, a Model Driven Engineering approach relying on SysML, and SoCLib, a SystemC based virtual prototyping platform.

3.1. AVATAR

The AVATAR environment [1] is a model-oriented solution for the analysis and design of embedded software. AVATAR relies on SysML diagrams to describe the software aspects of the system, as well as its safety and security properties. AVATAR is fully supported by the free software TTool [18]. With TTool, one can edit AVATAR models, simulate or verify them formally in a push-button approach. At last, just like in most UML approaches, simulation and formal verification rely on a purely timed functional model, i.e. without considering any hardware target (CPU, bus, etc.).

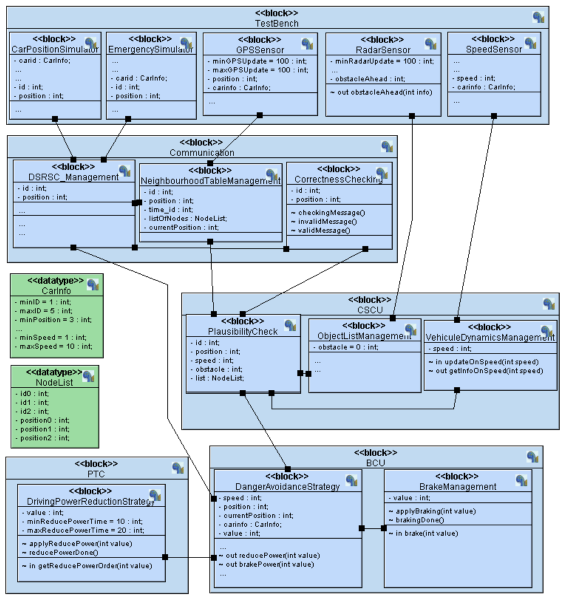
TTool implements an AVATAR-to-C/POSIX model transformation as shown in [2]. This code generation permits to validate the software models, but it does not offer any facility for Hardware/Software co-design or design space exploration. Said differently, hardware considerations cannot be captured in the original AVATAR model.

3.2. SoCLib

SoCLib [19] is a public domain library of component models written in SystemC. SoCLib targets shared-memory *multiprocessor-on-chip system* (MP-SoC) architectures based on the *Virtual Component Interconnect* (VCI) protocol [20] which separates the components' functionality from communication. SoCLib allows for timed TLM and cycle-accurate bit-accurate (CABA) simulation, so that we have a very detailed level of simulation, using instruction set simulators [21] and modeling cache behavior.

Design space explorations are also addressed in the scope of SoCLib, initially in the context of video streaming and telecommunication applications [22]. Mapping of software objects to memory banks is very fine-grained (stack, lock, buffers can be mapped separately if required), which is a significant improvement over tools like SPADE [6] where only functional tasks can be explicitly mapped.

SoCLib top cells are either hand-written, which is a cumbersome process, or generated from a Python specification, which are complex to describe and debug. Our approach combines both readability and ease of use, taking the latter one step further by proposing SysML/AVATAR as input format.



<<block>> TestBench
<<block>> CarPositionSimulator
<<block>> EmergencySimulator
<<block>> GPSSensor
<<block>> RadarSensor
<<block>> SpeedSensor
<<block>> Communication
<<block>> DSRSC_Management
<<block>> NeighbourhoodTableManagement
<<block>> CorrectnessChecking
<<datatype>> CarInfo
<<datatype>> NodeList
<<block>> CSCU
<<block>> PlausibilityCheck
<<block>> ObjectListManagement
<<block>> VehiculeDynamicsManagement
<<block>> PTC
<<block>> DrivingPowerReductionStrategy
<<block>> BCU
<<block>> DangerAvoidanceStrategy
<<block>> BrakeManagement

Figure 2. Block diagram of the Active Braking Use Case

4. Automotive Case Study

The AVATAR methodology is illustrated here by the same automotive embedded system designed in the scope of the European EVITA project [23] that was shown in [2]. Recent on-board Intelligent Transport (IT) architectures comprise a very heterogeneous landscape of communication network technologies (e.g., LIN, CAN, MOST, and FlexRay) that interconnect in-car Electronic Control Units (ECUs). The increasing number of such equipments - sometimes more than a

hundred - triggers the development of novel applications that are commonly spread among several ECUs to fulfill their goals.
The case study, an automatic braking application [24], works basically as follows: an obstacle is detected by another automotive system which broadcasts that information to neighboring cars. A car receiving such an information has to decide whether it is concerned with this obstacle, or not. This verification includes a plausibility check function that takes into account various parameters, such as the direction and speed of the car, and also information previously received from neighboring cars. Once the decision to brake has been taken, the braking order is forwarded to ECUs responsible for performing the emergency braking. Also, the presence of this obstacle is forwarded to other neighbor cars in case they have not yet received that information. Safety and security requirements were already given in [2].

Figure 2 represents the internal block diagram of the active braking use case. This internal block diagram comprises two kinds of blocks:

- **Blocks dedicated to the modeling of the environment**. They model messages received via wireless connections, data received from sensors, and data output to actuators. For instance, the block *CarPositionSimulator* models the car traffic around the considered automotive system. This car traffic generates location information to the system. The *GPSSensor* regularly records the car position.

- **Blocks dedicated to the modeling of the system itself**. Blocks are grouped within a parent block whose name is the one of the modeled ECU. Basically, the system model contains four ECUs: **Communication ECU** (receiving information, broadcasting information), **Chassis Safety Controller ECU (CSCU)**, **Braking Controller ECU (BCU)**, and **Power Train Controller ECU (PTC)**.

The *DrivingPowerReductionStrategy* block, part of the **Power Train Controller ECU (PTC)**, is not too complex to be shown on the limited space of this paper, and will serve as a running example. For our current experimentation, the security-related functions specified in the original application are left out, but not the safety ones, obviously. Also, the simulation and formal verification aspects have been presented in [2], which explains that the present paper focuses only on prototyping aspects.

5. Contribution

5.1. The AVATAR deployment diagram

Again, our main contribution is to enhance AVATAR with a hardware platform modeling capability, including the mapping of tasks and channels onto this platform. To do this, we rely on a newly defined **deployment diagram**, containing a SysML representation of hardware components, their interconnection, tasks and channels.

The AVATAR deployment diagram is this new modeling facility that allows a design to capture its hardware constraints. Figure 3 shows the main window of TTool with the deployment diagram of our example, the active braking application. We use a generic interconnect, a VGMN (Virtual generic Micro Network), along with five CPUs and one memory bank, which is not so realistic with regards to the real architecture, but far more convenient to explain the technical issues on code transformations. Platforms can be modified in a matter of minutes, for example by adding a second RAM and mapping part of the channels onto it. A valid platform must contain at least one CPU, one memory bank and one TTY.

5.2. New Tool Chain

From the deployment diagram, and from the previously existing software component diagrams, a fully executable SoCLib specification for a MPSoC can be generated. A new tool chain has been defined in order to support that model transformation (see Figure 4). The main components of this model transformation are:

- **libavatar** Runtime for SoCLib, implements the AVATAR operators.

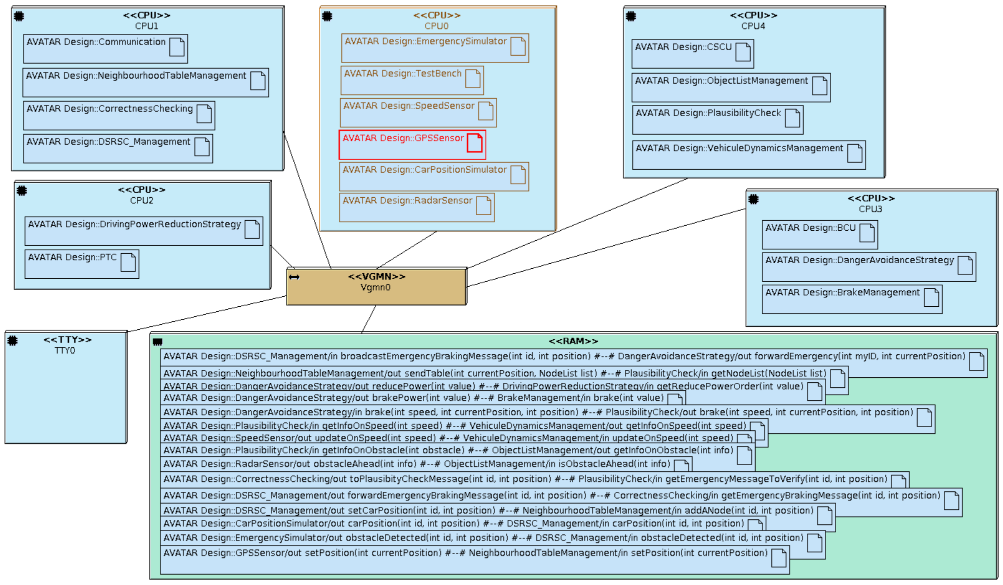
- **DDSyntaxChecker** checks the syntax of the deployment diagrams and identifies their elements.

- **AVATAR2SOCLIB** translates AVATAR blocks (i.e., software components) into C POSIX tasks and generates the main program.

- **TopcellGenerator** generates a SystemC top cell for cycle accurate bit accurate simulation.

- **LdscriptGenerator** generates the linker script taking into account the mapping specified in the deployment diagram.

Note that the arc crossing over between *AVATARDDSpecification* and *AVATAR2SOCLIB* is necessary: information from the deployment diagram is required to generate the application code.



<<CPU>>
CPU1
AVATAR Design::Communication
AVATAR Design::NeighbourhoodTableManagement
AVATAR Design::CorrectnessChecking
AVATAR Design::DSRSC_Management
<<CPU>>
CPU0
AVATAR Design::EmergencySimulator
AVATAR Design::TestBench
AVATAR Design::SpeedSensor
AVATAR Design::GPSSensor
AVATAR Design::CarPositionSimulator
AVATAR Design::RadarSensor
<<CPU>>
CPU4
AVATAR Design::CSCU
AVATAR Design::ObjectListManagement
AVATAR Design::PlausibilityCheck
AVATAR Design::VehiculeDynamicsManagement
<<CPU>>
CPU2
AVATAR Design::DrivingPowerReductionStrategy
AVATAR Design::PTC
<<CPU>>
CPU3
AVATAR Design::BCU
AVATAR Design::DangerAvoidanceStrategy
AVATAR Design::BrakeManagement
<<VGMN>>
Vgmn0
<<TTY>>
TTY0
<<RAM>>
AVATAR Design::DSRSC_Management/in broadcastEmergencyBrakingMessage(int id, int position) #--# DangerAvoidanceStrategy/out forwardEmergency(int myID, int currentPosition)
AVATAR Design::NeighbourhoodTableManagement/out sendTable(int currentPosition, NodeList list) #--# PlausibilityCheck/in getNodeList(NodeList list)
AVATAR Design::DangerAvoidanceStrategy/out reducePower(int value) #--# DrivingPowerReductionStrategy/in getReducePowerOrder(int value)
AVATAR Design::DangerAvoidanceStrategy/out brakePower(int value) #--# BrakeManagement/in brake(int value)
AVATAR Design::DangerAvoidanceStrategy/in brake(int speed, int currentPosition, int position) #--# PlausibilityCheck/out brake(int speed, int currentPosition, int position)
AVATAR Design::PlausibilityCheck/in getInfoOnSpeed(int speed) #--# VehiculeDynamicsManagement/out getInfoOnSpeed(int speed)
AVATAR Design::SpeedSensor/out updateOnSpeed(int speed) #--# VehiculeDynamicsManagement/in updateOnSpeed(int speed)
AVATAR Design::PlausibilityCheck/in getInfoOnObstacle(int obstacle) #--# ObjectListManagement/out getInfoOnObstacle(int info)
AVATAR Design::RadarSensor/out obstacleAhead(int info) #--# ObjectListManagement/in isObstacleAhead(int info)
AVATAR Design::CorrectnessChecking/out toPlausibityCheckMessage(int id, int position) #--# PlausibilityCheck/in getEmergencyMessageToVerify(int id, int position)
AVATAR Design::DSRSC_Management/out forwardEmergencyBrakingMessage(int id, int position) #--# CorrectnessChecking/in getEmergencyBrakingMessage(int id, int position)
AVATAR Design::DSRSC_Management/out setCarPosition(int id, int position) #--# NeighbourhoodTableManagement/in addANode(int id, int position)
AVATAR Design::CarPositionSimulator/out carPosition(int id, int position) #--# DSRSC_Management/in carPosition(int id, int position)
AVATAR Design::EmergencySimulator/out obstacleDetected(int id, int position) #--# DSRSC_Management/in obstacleDetected(int id, int position)
AVATAR Design::GPSSensor/out setPosition(int currentPosition) #--# NeighbourhoodTableManagement/in setPosition(int currentPosition)

Figure 3. Deployment Diagram of the Active Braking Application

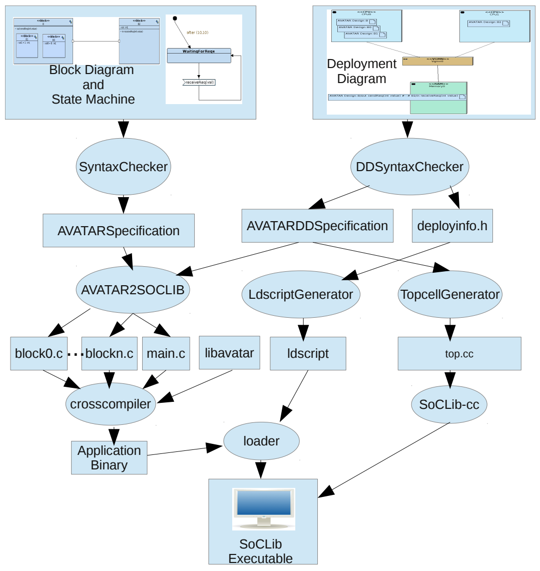


Figure 4. Tool chain

We now go through the main stages/elements involved in the model transformation.

5.3. The AVATAR Runtime ("libavatar")

The *AVATAR runtime* is a library of functions which capture the semantics of the AVATAR operators that appear in the code of the tasks (*delay, asyncRead*, etc.) and implements them using C/POSIX primitives. MutekH [25] is a free portable operating system for embedded platforms.

The AVATAR runtime more particularly focuses on channels, since the interconnect latencies and cache effects make the channels difficult to implement efficiently on a MPSoC platform. SoClib provides an efficient implementation of asynchronous channels as sowtware objects stored in on-chip memory, based on the Kahn [7] model, that can be accessed by any number of hardware or software reader and writer tasks alike [22]. Synchronous communications however require a central manager to resolve conflicts. In order to run on a SoCLib platform, instead of the local workstation, the runtime had to be adapted. Of course, the main model transformation issue is to maintain that precise semantics after transformation.

5.4. Code Generation

AvatarDeploymentPanelTranslator traverses the graphical elements of the DDiagram, extracts the data in form of objects (example: AvatarCPU) and adds it to a *AvatarDDSpecification object*.

5.4.1. Code Generation for AVATAR Blocks. Each mapped block is translated into a C POSIX thread. As an example, Figure 5 shows the state machine of the *DrivingPowerReductionStrategy* block. On receiving a signal *getReducePowerOrder* in the *WaitForReducePowerOrder* state, a waiting time within a given interval (minimum, maximum) is taken into consideration before the order is executed (state *WaitForReducePowerToBePerformed*). Figure 6 shows the generated code for

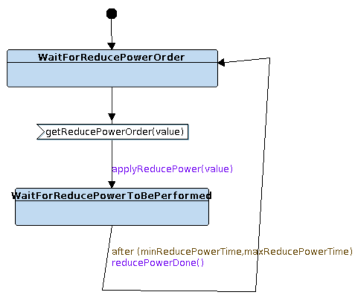

WaitForReducePowerOrder
getReducePowerOrder(value)
applyReducePower(value)
WaitForReducePowerToBePerformed
after (minReducePowerTime,maxReducePowerTime)
reducePowerDone()

Figure 5. State Machine of the Block DrivingPowerReductionStrategy

the *DrivingPowerReductionStrategy* block. The state machine has three states, including the start state. State *WaitForReducePowerOrder* awaits a synchronous message, which has been enqueued in a list of pending requests of the overall system by another block named *DangerAvoidanceStrategy* (see lower left of Figure 2). Then, *WaitForReducePowerToBePerformed* waits for a delay randomly selected between 10 and 20 milliseconds. That delays corresponds to the actuators work to really reduce the power.

```
#include "DrivingPowerReductionStrategy.h"
static uint32_t _getReducePowerOrder;

#define STATE__START__STATE 0
#define STATE__WaitForReducePowerToBePerformed 1
#define STATE__WaitForReducePowerOrder 2
#define STATE__STOP__STATE 3
...
void *mainFunc__DrivingPowerReductionStrategy(void *arg){
  int value = 0;
  int minReducePowerTime = 10;
  int maxReducePowerTime = 20;

  int __currentState = STATE__START__STATE;
  ...
  pthread_cond_init(&__myCond, NULL);
  fillListOfRequests(&__list, __myname, &__myCond, &__mainMutex);

  while(__currentState != STATE__STOP__STATE) {
    switch(__currentState) {
      case STATE__START__STATE:
      __currentState = STATE__WaitForReducePowerOrder;
      break;

      case STATE__WaitForReducePowerOrder:
      __params0[0] = &value;
      makeNewRequest(&__req0, 853, RECEIVE_SYNC_REQUEST,0,0,0,1, __params0);
      __req0.syncChannel = &__DangerAvoidanceStrategy_reducePower\
__DrivingPowerReductionStrategy_getReducePowerOrder;
      __returnRequest = executeOneRequest(&__list, &__req0);
      clearListOfRequests(&__list);
      DrivingPowerReductionStrategy__applyReducePower(value);
      __currentState = STATE__WaitForReducePowerToBePerformed;
      break;

      case STATE__WaitForReducePowerToBePerformed:
      waitFor(minReducePowerTime, maxReducePowerTime);
      DrivingPowerReductionStrategy__reducePowerDone();
      __currentState = STATE__WaitForReducePowerOrder;
      break;
    }
  }
  return NULL;
}
```

Figure 6. Extract from the generated code for the DrivingPowerReductionStrategy block

5.4.2. Main Program Generation. The main program instanciates all necessary elements, e.g. the POSIX threads of the AVATAR blocks, and the SoCLib channels translated as software objects stored in the on-chip memory: these channels correspond to the AVATAR channels. Threads, corresponding to an AVATAR block each, are spawned from the main thread. Via *pthread_attr*, they are forced onto the CPU indicated in the Deployment Diagram. For example, *attr_t->cpucount=2* forces thread *DrivingPowerReductionnStrategy* onto CPU2. In order to map channels to specific memory areas, we name a software object whose mapping will be performed in the linker script using the same identifier.

Figure 7 shows an extract from the main program, focusing on our example block. First, the POSIX threads are initialized. Next, the signals belonging to the channel are associated to their input and output ports (for lack of space, we show this only for the channel *DrivingPowerReductionStrategy_getReducePowerOrder*). The threads are then created, attributes set beforehand, for example *attr_t->cpucount* to force a thread upon CPU 2. Finally, all threads are joined to wait for the program completion.

```
/* Synchronous channels */
syncchannel __DangerAvoidanceStrategy_reducePower\
__DrivingPowerReductionStrategy_getReducePowerOrder;
...
int main(int argc, char *argv[]) {
  void *ptr;
  pthread_barrier_init(&barrier,NULL, NB_PROC);
  pthread_attr_t *attr_t = malloc(sizeof(pthread_attr_t));
  pthread_attr_init(attr_t);
  pthread_mutex_init(&__mainMutex, NULL);

  /* Synchronous channels */
  __DangerAvoidanceStrategy_reducePower\
__DrivingPowerReductionStrategy_getReducePowerOrder.inname
    ="getReducePowerOrder";
  __DangerAvoidanceStrategy_reducePower\
__DrivingPowerReductionStrategy_getReducePowerOrder.outname
    ="reducePower";
  ...
  /* Threads of tasks */
  pthread_t thread__DrivingPowerReductionStrategy;
  ...
  ptr =malloc(sizeof(pthread_t));
  thread__DrivingPowerReductionStrategy= (pthread_t)ptr;
  attr_t = malloc(sizeof(pthread_attr_t));
  attr_t->cpucount = 2;

  pthread_create(&thread__DrivingPowerReductionStrategy, NULL,
    mainFunc__DrivingPowerReductionStrategy, NULL);
  ...
  pthread_join(thread__DrivingPowerReductionStrategy, NULL);
  return 0;
}
```

Figure 7. Extract from generated main program

5.5. Top Cell Generation

SoCLib is based on the *shared memory* paradigm, where a target is identified by the most significant bits of its address in a common memory space. The top cell generator thus must determine, for each target component, its unique target number associated to the segment address. We opted for the generic platform described in Figure 8: the platform considered in the case study has been derived from this generic platform. The platform features five PowerPC cores with integrated data and instruction cache (Xcache), one TTY, and one memory bank. Components are interconnected by a VGMN (see Section 5.1). VCI target interfaces are depicted with lighter arrows, initiators with darker arrows.

Some features must be explicitly captured in the Deployment Diagram, like CPUs and memory banks, as shown in Figure 3, while others are totally hidden to the TTool user, e.g. the numbering of target segments. Moreover, a timer, ICU and simhelper containing some simulation support facilities are currently generated *transparently* for the user. In the same way, the size of memory segments is given by a default value and the starting addresses are calculated by the top cell generator tool. Specific use cases may need other formats and must therefore modify the generated code.

The mapping of tasks to processors has no impact on the top cell, the sections containing channels however have to be listed in the call to the loader, see upper part of Figure 9.

The top cell generation takes as input the *AvatarDDSpecification* object (see Figure 4), and proceeds as follows:

1. We generate all segment addresses transparently, with exception of the segments containing channels which are visible in the Deployment Diagram.

2. All platforms use a *flattened device tree* (FDT) stored in ROM and contain a *simhelper* component for simulation support.

3. We systematically add a multi-timer and an interrupt control unit (ICU) target, even if not used by the application, and generate connection of all possible interrupt lines.

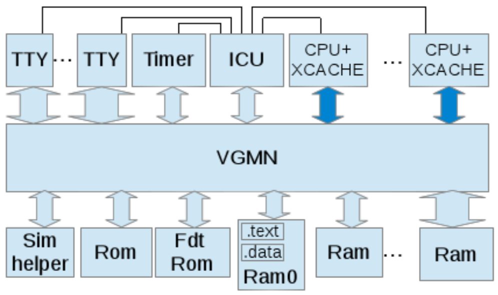


Figure 8. Generic SoCLib Platform

Top cell

```
data_ldr.load_file(std::string(kernel_p)
 + ";.data;.channel0;  ...
      .channel14;.cpudata;.contextdata");
```

Excerpt of the *ldscript*
Mapping channels on a single RAM:

```
.channel0  : { *(section_channel0)} > mem_ram
...
.channel14 : { *(section_channel14)} > mem_ram
```

Mapping on two RAMs:

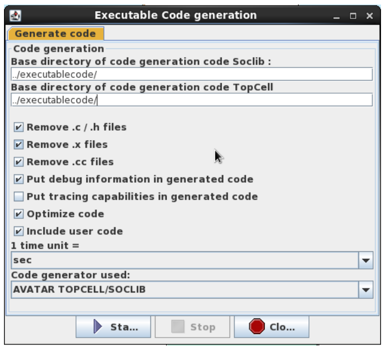
```
.channel0  : { *(section_channel0)} > mem_ram0
...
.channel14 : { *(section_channel14)} > mem_ram1
```

Figure 9. Top cell (top) and ldscript (bottom)

5.6. The Linker Script Generator

Handling the mapping of channels is rather difficult since it is related to the main program, the top cell and the ldscript. The linker script (*ldscript*) is a file which defines the memory layout; it associates each entry section to an output section, which receives its address in memory. It is generated by a tool developed in the context of [2] and uses essentially the C preprocessor.

The mapping information of channels in *AvatarDeploymentPanelTranslator* are then used to generate a file *deployinfo.h*: the latter is meant to be included by the ldscript generator. the lower part of Figure 9 shows an excerpt from the generated ldscript for our example. Thus, we first map all channels on one memory bank, then on two different banks. The section *.channel* of



Code generation
Base directory of code generation code Soclib : ../executablecode/
Base directory of code generation code TopCell ../executablecode/

- [✓] Remove .c / .h files
- [✓] Remove .x files
- [✓] Remove .cc files
- [✓] Put debug information in generated code
- [ ] Put tracing capabilities in generated code
- [✓] Optimize code
- [✓] Include user code

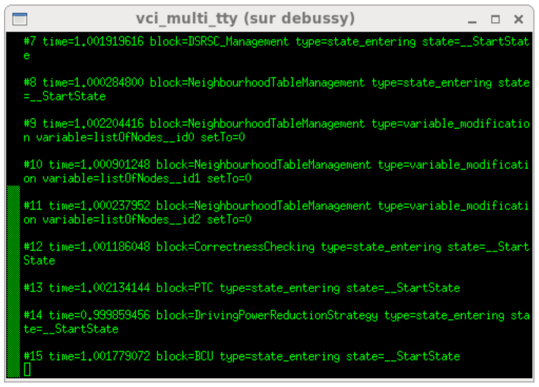
1 time unit = sec
Code generator used: AVATAR TOPCELL/SOCLIB

Figure 10. Menu for SoCLib code generation

the *ldscript* is made from *section_channel* which in turn contains the software object specified in the main program in order to represent the channel. Several channels can be placed on the same memory bank. The loader invoked in the top cell (lower part of figure 9 then places it on the RAM as specified in the Deployment Diagram.

5.7. Using TTool/SoCLib

The generation of the code, top cell and ldscript has been integrated into TTool with a press-button approach. When checking the syntax of the deployment diagram, TTool also now checks for related blocks, interconnections between blocks, and state machines diagrams. Then, TTool displays the code generation dialog window from which executable code can now be generated, as explained in previous section, for the SoCLib platform. Figure 10 shows the code generation menu of TTool. Figure 11 shows the resulting SoCLib simulation. The application and main program code generation takes less than one second. The generation of the SystemC executable takes around 30 seconds on a 64 bit 4 core Xeon 2 Gbyte RAM machine under scientific Linux 64 bit (Table 1). Among others, CABA level simulation potentially allows to measure cache miss rates, latency of each step of the individual cache miss, traffic on the interconnect, latency of each transaction on the interconnect, fill state of the buffers, knowing which lock is taken/released and the cycle when this happens. For example, the effect of a remapping of channels as shown in the upper part of Figure 9, or a change to the cache parameters can be analyzed in full detail.



```
#7 time=1.001919616 block=DSRSC_Management type=state_entering state=__StartStat
e

#8 time=1.000284800 block=NeighbourhoodTableManagement type=state_entering state
=__StartState

#9 time=1.002204416 block=NeighbourhoodTableManagement type=variable_modificatio
n variable=listOfNodes__id0 setTo=0

#10 time=1.000901248 block=NeighbourhoodTableManagement type=variable_modificati
on variable=listOfNodes__id1 setTo=0

#11 time=1.000237952 block=NeighbourhoodTableManagement type=variable_modificati
on variable=listOfNodes__id2 setTo=0

#12 time=1.001186048 block=CorrectnessChecking type=state_entering state=__Start
State

#13 time=1.002134144 block=PTC type=state_entering state=__StartState

#14 time=0.999859456 block=DrivingPowerReductionStrategy type=state_entering sta
te=__StartState

#15 time=1.001779072 block=BCU type=state_entering state=__StartState
```

Figure 11. Using SoCLib from TTool

top cell and ldscript generation	main and block code generation	compilation to application executable	soclib platform compilation
0.31 s	1.14 s	14.61 s	28.74 s

Table 1. Performance results for the case study

6. Current Limitations

The approach is already available in TTool an experimental branch (-experimental option). Yet, it will be publicly available before the ERTS'2016 edition (a live demonstration will be performed).

From the model and translation point of view, synchronous channels currently require the use of a central manager, thus generating a significant overhead due to synchronization traffic on the interconnect. Adding a specific support for specific automotive interconnects (CAN, flexray) is planned.

From the simulation perspective, we opted for a prototyping environment with a low-level abstraction level (CABA level). Thus, an obvious limitation is the simulation speed. We need to get comparative results and work on speeding up the simulation. The simulation speed drawback is probably due to other factors. Currently, we use a flat, generic interconnect, however with contention and cache effects. Also, instead of a detailed CABA model of the processor, we use a instruction set simulator [21] to speed up simulation. Due to simulation complexity, we are still limited to some dozen processors, yet, it should be enough for most embedded applications.

A variety of low level performance measuring tools exists for SoCLib, among others giving cycle level information on the channel fill state and latency on the interconnect [26, 27]; these will have to be integrated, opening the way to automated feedback of performance results and, ultimately, Design Space Exploration.

7. Discussion and Future Work

The paper demonstrates the use of a recent extension to the TTool/AVATAR environment, by a larger case study stemming from automotive systems. One strength of our approach is that it offers a prototyping and exploration solution for engineers from industry, accustomed to the use of UML/SysML diagrams, while maintaining precise simulation results in addition to formal proofs, all in a joint framework. Our framework targets embedded systems with complex platforms, such as the ones found in telecommunication and transportation applications, but also automotive [23] and avionics providers, in particular those who already use AVATAR.

The next technical steps will consist in supporting more hardware components (e.g., co-processor wrappers, other types of interconnect, DMA), and more mapping capabilities (e.g., stacks, locks, see [22]). The support of synchronous channels requires a central request manager as such a semantics is not natively supported by SoCLib. Also, the current trade-off between simplicity and functionality might be reconsidered w.r.t. the usage of this new prototyping environment.

In TTool, the animation of the model in the prototyping phase is currently limited to a sequence diagram displaying transactions between software components. It would also be useful to provide information about hardware nodes, e.g., the load of processors, the state of buffers and the traffic on the interconnection network during simulation. Last by not least, for the moment, design space exploration is done by hand, The roadmap of our project envisages to integrate the feedback from detailed simulation such that it can be taken into account by the high level models.

Acknowledgments We would like to thank Alexandre Becoulet for his assistance on MutekH.

References

[1] G. Pedroza, D. Knorreck, and L. Apvrille, "AVATAR: A SysML environment for the formal verification of safety and security properties," in *The 11th IEEE Conference on Distributed Systems and New Technologies (NOTERE'2011)*, Paris, France, May 2011.

[2] L. Apvrille and A. Becoulet, "Prototyping an embedded automotive system from its UML/SysML models," in *ERTSS'2012*, Toulouse, Feb. 2012.

[3] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: a framework for simulating and prototyping heterogeneous systems," *Readings in hardware/software co-design*, pp. 527–543, 2002.

[4] P. Lieverse, P. van der Wolf, K. A. Vissers, and E. F. Deprettere, "A methodology for architecture exploration of heterogeneous signal processing systems," *VLSI Signal Processing*, vol. 29, no. 3, pp. 197–207, 2001.

[5] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. L. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *IEEE Computer*, vol. 36, no. 4, pp. 45–52, 2003.

[6] P. Lieverse, T. Stefanov, P. van der Wolf, and E. F. Deprettere, "System level design with spade: an M-JPEG case study," in *ICCAD*, 2001, pp. 31–38.

[7] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing '74: Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed. New York, NY: North-Holland, 1974, pp. 471–475.

[8] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 358–374, 2006.

[9] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, and E. F. Deprettere, "Exploring embedded-systems architectures with artemis," *IEEE Computer*, vol. 34, no. 11, pp. 57–63, 2001.

[10] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with UML and MARTE," in *DATE'09*, April 2009, pp. 226–231.

[11] A. Gamatié, S. L. Beux, É. Piel, R. B. Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser, "A model-driven design framework for massively parallel embedded systems," *ACM Trans. Embedded Comput. Syst*, vol. 10, no. 4, p. 39, 2011.

[12] Sodius Corporation, "Mdgen for SystemC," http://sodius.com/products-overview/systemc.

[13] P. H. Feiler, B. A. Lewis, S. Vestal, and E. Colbert, "An overview of the SAE architecture analysis & design language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering," in *IFIP-WADL*, ser. IFIP, P. Dissaux, M. Filali-Amine, P. Michel, and F. Vernadat, Eds., vol. 176. Springer, 2004, pp. 3–15.

[14] M. Ben Youssef, J.-F. Boland, G. Nicolescu, G. Bois, and J. Hugues, "Bridging the high-level model to execution platform for design space exploration and implementation," in *ERTSS*, 2014.

[15] Polarsys, "ARCADIA/CAPELLA (webpage)," in *https://www.polarsys.org/capella/arcadia.html*, 2008.

[16] J. Bengtsson and W. Yi., "Timed automata: Semantics, algorithms and tools," in *Lecture Notes on Concurrency and Petri Nets*. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

[17] B. Blanchet, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, Jul. 2009.

[18] L. Apvrille, "Webpage of TTool," in *http://ttool.telecom-paristech.fr/*, 2015.

[19] SoCLib consortium, "SoCLib: an open platform for virtual prototyping of multi-processors system on chip (webpage)," in *http://www.soclib.fr*, 2010.

[20] VSI Alliance, "Virtual Component Interface Standard (OCB 2 2.0)," VSI Alliance, Tech. Rep., Aug. 2000.

[21] N. Pouillon, A. Becoulet, A. V. de Mello, F. Pêcheux, and A. Greiner, "A generic instruction set simulator API for timed and untimed simulation and debug of MP2-socs," in *RSP*. IEEE, 2009, pp. 116–122.

[22] D. Genius, E. Faure, and N. Pouillon, "Mapping a telecommunication application on a multiprocessor system-on-chip," in *Algorithm-Architecture Matching for Signal and Image Processing*, G. Gogniat, D. Milojevic, and A. M. A. A. Erdogan, Eds. Springer LNEE vol. 73, Nov. 2011, ch. 1, pp. 53–77.

[23] EVITA, "E-safety Vehicle InTrusion protected Applications," http://www.evita-project.org/.

[24] E. Kelling, M. Friedewald, T. Leimbach, M. Menzel, P. Sger, H. Seudié, and B. Weyl, "Specification and evaluation of e-security relevant use cases," EVITA Project, Tech. Rep. Deliverable D2.1, 2009.

[25] A. Becoulet, "Mutekh," http://www.mutekh.org.

[26] D. Genius and N. Pouillon, "Monitoring communication channels on a shared memory multi-processor system on chip," in *ReCoSoC*. IEEE, 2011, pp. 1–8.

[27] D. Genius, "Measuring Memory Latency for Software Objects in a NUMA System-on-Chip Architecture," ReCoSoc, Darmstadt, Germany, Jul. 2013.

Linking model predictive control (MPC) and system simulation tools to support automotive system architecture choices

Dirk von Wissel, Vincent Talon (RENAULT)
Vincent Thomas, Benoist Grangier (SIEMENS)
Lukas Lansky, Michael Uchanski (HONEYWELL)

Abstract

Today's automotive industry must introduce advanced powertrain technologies as a consequence of the stringent environmental regulations and strong market expectations. This leads to the increase of vehicle variants and to the growth of the powertrain architectures complexity. Hence, the development of hardware and software can no longer be decoupled. Those activities have to run in parallel, starting from simulation and advanced engineering, continuing to the detailed engineering phase, and ending in validation and calibration. In the so-called Model-Based Systems Engineering (MBSE) approach, simulation models including both hardware and control systems are used to first make decisions on possible architectures. This paper presents a new MBSE approach that allows powertrain hardware selection to occur early during the simulation stages. The process combines Model Predictive Control (MPC) for the control system with a physical modeling software package for the hardware. This combination of MPC and physical modeling addresses several practical difficulties that typically hinder attempts at MBSE for hardware selection.

1. Introduction

Context

Today's automotive powertrains must delight customers with performance, reliability, and low noise while simultaneously meeting increasingly stringent regulations on CO_2 and other emissions, all at a competitive cost and fast time-to-market. Embedded software, electronics, sensors and actuators play a critical role and acts as a "glue" to make combustion systems, boosting systems, cooling systems, exhaust after-treatment systems, batteries, and transmissions perfectly integrate for optimal vehicle performance.

The development of the mechanical powertrain hardware and the software functions controlling them can no longer be performed independently. As a result, many automakers implement hardware selection processes where many powertrain variants are built. An electronic control system is then designed and calibrated for each variant. The variant that best meets system-level requirements moves to the next development stage. But this process is money and time-consuming and prevents manufacturers and suppliers to stay in the competition race.

Given the financial incentives, most automotive companies have tried some form of Model-Based-System-Engineering for hardware selection applying system simulation in particular. This implies to have access to the "right" models for both the hardware and the software.

Difficulties frequently encountered with MBSE for hardware selection

Models for the plant (engine, transmission, and vehicle) are generally available at the earlier stages of the product development process. Often generated during R&D and advanced projects, they are not necessarily well adapted to be applied for an innovative and efficient hardware selection process. In particular, the models generated in the with a powertrain subsystem design perspective are often too complex and slow to be combined efficiently with control tools. The present paper illustrates the requirements for getting the right modeling approach for plant model adapted to hardware selection and the implications for the deployment of models throughout the complete product development process.

On the other hand, when models of the plant are available, obtaining the corresponding control models is always a more difficult task. One of two methods is usually employed:

- Adaptation of existing production control approaches – This approach provides a path to the implementation on physical prototypes later in the process. However, simulation time can be slow because the production strategy block diagrams often include tens of thousands of elements. Personnel gaps are an even more serious issue. Hardware designers are usually experts in boosting, transmissions, or system-level simulation tools, but have limited skills in complex control strategy diagrams.

- Simplified custom controllers – A simplified control is created from scratch to cover the extra capabilities and degrees of freedom required for the new technology brick. This approach offers fast simulation times and understandable control models. But also come with disadvantages: the time and expertise required to build the controller from scratch is often substantial. Also, since it is often built quickly, controller performance can often be substantially different from that of a well-tuned production controller. As a result, controller tuning differences could be mistaken for hardware performance differences. Finally, the controller used in simulation is often a dead-end with no path to implementation on embedded computing platforms.

Novelty of the proposed approach

The process described in this paper improves on the second method by making it faster to develop custom controllers for new hardware: The novelty is to use physical models with MPC (Model Predictive Control), a systematic, easy-to-use requirement based controller design methodology. The focus of the approach is put on ease. Both Model Based controller Design (MBD) and optimal control are well established in the field of controls engineering but are under-applied in automotive due to the specialized knowledge required to apply them. A dedicated, easy-to-use MPC software package helps to overcome this difficulty and offers the benefits of model based and optimal control design to a larger community of engineers.

The tool approach also offers the possibility to port MPC directly from numeric simulation to embedded software, which can actually run on physical hardware such as Electronic Control Units. Finally, as MPC yields nearly optimal control, tuning is generally of good quality, making it less likely that control tuning choices will mask differences between mechanical hardware.

Methodology

This paper presents a new Model-Based Systems Engineering (MBSE) approach based on the application of existing tools interfaced in order to generate a fast and efficient workflow that allows powertrain hardware selection. The approach is structured around four main stages:

- Development of a non-linear physical plant model for the studied system
- Processing of the model in order collect linear models applied for the design of a MPC controller
- Validation of the controller using a Model-In-the-Loop environment including the baseline plant model
- Evaluation of the system potential using the plant with controller regarding relevant criteria

The present paper illustrates how this combination of MPC and physical modeling addresses several practical difficulties that typically hinder attempts at MBSE for hardware selection.

The application and benefits of the approach are illustrated on a use case from RENAULT with the benchmarking analysis of gasoline air path systems using a tool chain including LMS Imagine.Lab Amesim combined with the Honeywell OnRAMP Design Suite.

2. Application of system simulation (Existing tool)

Commercial CAE software is a critical lever toward the management of system complexity and limited development times. This is why it is widely deployed in automotive industry from the early stages of development to the final stages before SOP, where system simulation is applied to validate system integration.

The hardware selection process in the early stages of an innovative product development process can take advantage of the existing deployment of CAE environments and more importantly, its usage for system simulation. The capability to evaluate numerous technical options in a limited time gives a competitive advantage compared to conventional iterative process with real prototypes.

General requirements for system simulation

The application of a simulation based decision process also raises three major challenges in terms of tooling capabilities:

- Integration in a larger MBSE process
  System hardware selection is located early in the development process. In this so called system pre-design phase all system component specifications have to be detailed. The system hardware selection process must be supported by models generated upstream in the V-cycle, which can include an additional level of complexity linked to the poor availability of data. The direct application of high fidelity plant modeling (1D software) is often not appropriate as this can jeopardize the trade-off between prediction capabilities and run times.

- Diversity of hardware selection topics
  One of the challenges for the system simulation software is to offer the capability a high level of versatility to address different levels of modeling to fit with the variety of systems and sub-systems that must be investigated by car manufacturers. This is why a high level of flexibility and scalability is required from simulation software. For example in the selection of the right air path architecture for a given engine the details of the intake and exhaust flows and their impact on the combustion process have to be known in a precise way while the rest of the vehicle can be modeled in a simple way to study vehicle attributes like fuel consumption or performance. Another example is the development of a hybrid powertrain in which all related powertrain subsystems have to be detailed with high precision but in which the air path system can be described in a simple way.

- Interfacing with control development environments
  Since hardware models must ultimately be combined with the software model, a common approach is to apply co-simulation technics. Most of the simulation tools offer this kind of interface with Simulink or even with C codes. The problem with co-simulation is that it often has an over-cost in terms of run times. That is also in favor of the application of 0D models compared to high fidelity 1D modeling approaches.

Application with LMS Imagine.Lab Amesim

The 0D models in the LMS Imagine.Lab Amesim (LMS Amesim) software package provide sufficient fidelity to be used in hardware development while remaining simple enough for its use in control design. The 0D lumped parameter approach, which is used in LMS Amesim, offers a good compromise between high level vehicle simulation tools, which are often too simple, and full 1D software that is often too complex. High level software are able to predict the full vehicle attributes over driving cycle but do not allow going deeper into sub-systems details. 1D software is well-suited to engine design, but runs slowly when linked to control environments.

Taking advantage of this flexibility of the LMS Amesim software, RENAULT has progressively adopted it as standard tool for system simulation and plant modeling. It is currently deployed for advanced engineering, vehicle planning phase and supports the full control development cycle from design to MIL to HIL. RENAULT's large plant model database includes detailed engine models, transmissions, actuator models, and a full hybrid vehicle model, and is updated continuously with each new vehicle or powertrain program. RENAULT's controls engineers use these models for system simulation and software validation and calibration

LMS Amesim's scripting capabilities and its ability to automatically produce a linearization of full plant models are of special interest for the hardware selection process described in this paper. Both features were used to establish a connection to the MPC control design suite HONEYWELL OnRAMP. On one hand, LMS Amesim is an open software that is delivered with scripting APIs (Matlab, Python…) that ease its integration in processes and application oriented workflows. Hence, scripts were developed in order to pilot remotely the LMS Amesim core, to get the plant model description (definition of I/O, units, ports…), to gather a linear model for a given operating point and to launch non-linear simulations to check the validity of the linear model and validate the designed control function. On the other hand, the 0D software package is able to extract from a physical non-linear model a linear model at a given operating point, which is possible thanks to the use of ordinary differential equations (ODE). Indeed, the numerical methods applied to solve physical equations in CFD1D or 3D software do not support this feature. The automatic creation of linear models from an existing non-linear 0D model is the inbuilt LMS Amesim core capability that made it possible to create the interface with the HONEYWELL MPC design software.

3. User-friendly MPC control design tool (Existing tool)

MPC (Model Predictive Controls) has been a preferred controls methodology in process industries for several decades thanks to its rapid, systematic approach. Over the past decade, its use has significantly increased in the automotive controls community due to theoretical advances which reduce processing time, and due to the increased computing power of the ECUs themselves. At RENAULT several powertrain control innovation projects are already using MPC in a quasi-industrial context. Other automotive OEMs are also active applying MPC to automotive problems including traction control, vehicle stability control, and exhaust system control.

For the MPC part of this work, RENAULT selected the HONEYWELL OnRAMP software toolset. OnRAMP™ Design Suite is a software tool for modelling and design of advanced control algorithms for a wide range of automotive applications. There are many aspects to consider when evaluating such a tool: certification, compatibility, controller performance, footprint in the ECUs, calibration and documentation. OnRAMP addresses these as a package.

Advanced control toolboxes from academia or the commercial sector often offer significant flexibility and advanced features for MPC design. However they often require advanced knowledge from the user and offer limited support for wide spread deployment within an organization. OnRAMP offers a GUI-driven workflow and automatic tuning algorithms to facilitate the design, tuning and performance evaluation of the MPC controller. The OnRAMP workflow delivers C code that run on more than 15 OEM target environments.

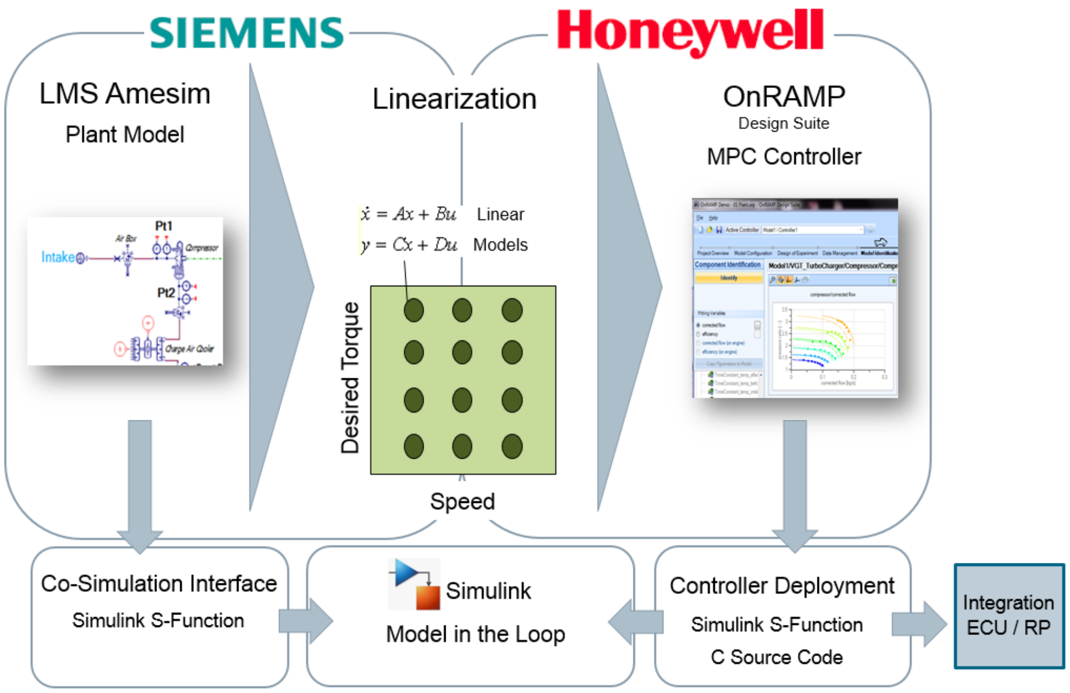
Thus, OnRAMP is targeted at automakers and suppliers who would like to benefit from MPC by making it fast and easy to use for a wide audience including engine calibrators, controls generalists, and MPC controls specialists. The tool proposes a systematic workflow and automates model building and tuning tasks, which are tedious, or that require specialized MPC knowledge.

OnRAMP includes its own modeling libraries. However, the present work seeks to leverage RENAULT's large database of existing physical plant models in LMS Amesim format.

4. Combining the MPC and system simulation tools into a new MBSE workflow

Since the missing piece of MBSE for powertrain hardware selection is a fast, convenient controller synthesis, it seems natural to link a system simulation tool like LMS Amesim with a MPC control synthesis tool like OnRAMP. RENAULT engineers have championed this work because they believe that MPC can fill the controls gap in MBSE.

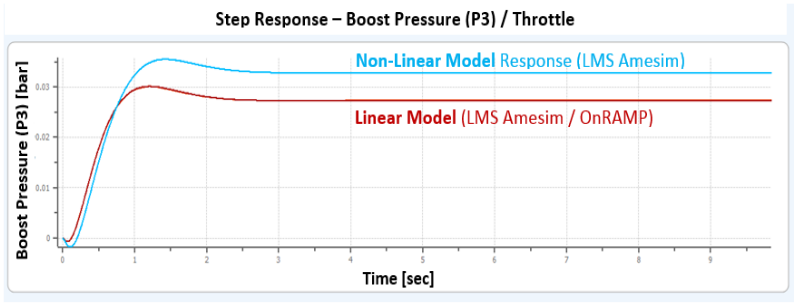
As aforementioned, linking the two tools together is enabled by, first linearization capabilities and secondly by the scripting features of the Siemens PLM Software system simulation platform. The Figure below illustrates the process applied for the design up to the validation of the MPC controller.



The first step of the process is to identify inputs, outputs, disturbances and scheduling variables to the controller. This is determined thanks to a modeling convention in LMS Amesim. In practice, the interface between the two tools is concretized by an interface block included in the plant model which defines the actual control I/O.

The second step of the process is steady-state model analysis and feed-forward design. OnRAMP sends, through the API, actuator positions to the LMS Amesim model with the objective of controlling the plant to satisfactory steady state operating point. LMS Amesim responds with the plant's steady state response, as well as with linear state space models (dx/dt = Ax + Bu, y = Cx + Du) that approximate the behavior of the plant at the proposed actuator positions. OnRAMP then adapts its proposed actuator positions, using information from the A, B, C and D matrices to search in the right direction. Consequently the feedforward actuator positions are found achieving desired set-points while respecting constraints. The non-linear model is then linearized at these feed-forward actuator positions by LMS Amesim. The generated linear models are then processed further in OnRAMP for scaling, order reduction and discretization.

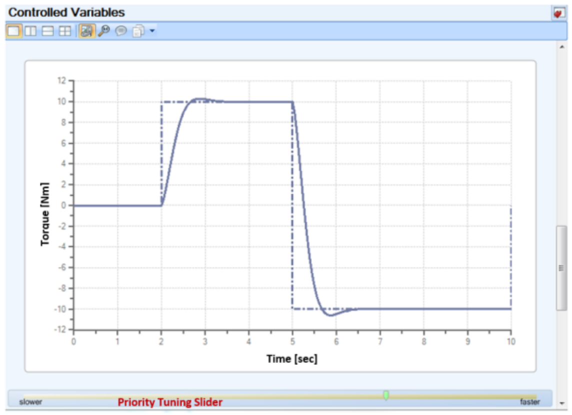
Furthermore the non-linear and linear models step responses are visualized and checked within OnRAMP. This is actually one of the key steps for successful design of MPC feedback control later. Figure on the right illustrates this.



The third step is scheduled linear MPC feedback design. The standard MPC cost function consists of penalties for actuator movement, tracking error and soft constraints. This is outlined in the optimization problem shown in the equation below.

$$\min_{U_{N_u};\,\varepsilon_1} J(U; x(k); v(k)) = \sum_{j=0}^{N_y} \|e(k+j|k)\|_2^Q + \|\delta u(k+j)\|_2^R + \|\rho_1 \cdot \varepsilon_1\|_2^2$$

Important part of feedback control design within OnRAMP is automatic tuning algorithm to achieve user defined requirements on robustness and bandwidth. This is an essential feature to enable generic design for a wide variety of plants. The user requirements are used to formulate the robust stability condition and OnRAMP then manipulates weighting matrices (e.g. for Q and R in the above) of the MPC cost function such that this condition is satisfied. The controller tuning then consists mainly of adjusting controller bandwidth, relative weights of controlled and manipulated variables and choice of prediction horizon. This leads together with earlier model checks to systematic and intuitive design and tuning of a MPC feedback control.



Finally, the controller designed within OnRAMP is tested in a closed-loop environment, using co-simulation methods. Indeed, the OnRAMP tool can generate the C code for the control and the associated data that can be either exported in Simulink. At this stage, the MiL environment is define as a co-simulation between LMS Amesim and Simulink or can be set in the Siemens platform by importing the Simulink model directly into the LMS Amesim GUI.

5. Application to hardware selection – engine air path systems

RENAULT engine portfolio already includes gasoline engines that comply with the Euro 6 emissions standard. Engineers are already in the process of upgrading current engines to fit with the upcoming regulations involving new driving cycles like the Worldwide harmonized Light vehicles Test Procedures (WLTP) and Real Driving Emissions (RDE). An update of the control logic and calibration will not be sufficient to fulfill these requirements. As a consequence, new mechanical hardware must be selected.

RENAULT selected a set of candidate suppliers for 10 candidate hardware components, or “technology bricks.” The standard implementation would require to build 10 prototypes and to run test programs to benchmark every component. The resulting costs and time would not be acceptable for RENAULT.

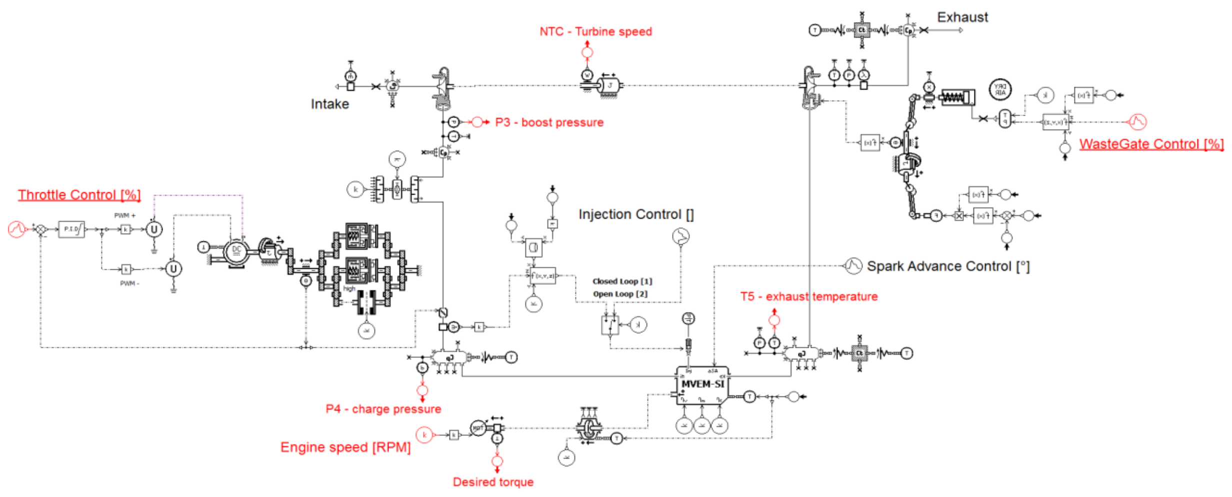
As a consequence, RENAULT decided to apply MBSE and to build 10 virtual prototypes to determine the optimal combination of technology bricks while keeping the associated costs under control. This required building 10 Model-In-the-Loop platforms coupling the plant models with their control models. This can be seen as a very complex, time-consuming task requiring highly skilled engineers for both the plant model and control model development or adaptation. Actually, the process applied by RENAULT does permit to tackle these challenges by taking advantage of:

- LMS Amesim with its well-balanced 0D modeling approach,
- OnRAMP with its fast and systematic control design workflow
- The coupling of two software one for the design of the other for the controller design and the set-up of the MIL environment

Illustration of the application of the tool chain

The workflow detailed in the paper is illustrated on a simple use case related to the evaluation of hardware for a gasoline air path system. Indeed, downsized turbocharged gasoline engine attributes are strongly dependent on the air path and charging technologies and the associated control. Many different hardware and related architectures are proposed by suppliers, from the common waste-gated turbocharger, to electrical driven compressor and dual-stage devices and the engine manufacturer needs to evaluate their technical versus cost potential.

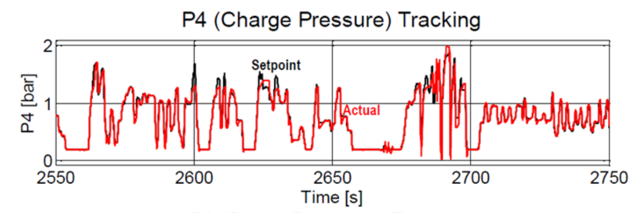
As a starting point, the tool chain is applied to a standard air path configuration including a turbocharger with a waste-gate and a throttle valve. A physical model for this baseline engine is developed in the LMS Amesim platform, its inputs and outputs are properly set according to the control problem and for the interfacing with OnRAMP. In practice, since the purpose of this engineering task is to define the potential of the system for steady-state and transient operation, models for the actuators were also included in the model as below so as to get an accurate prediction of the real system time response.

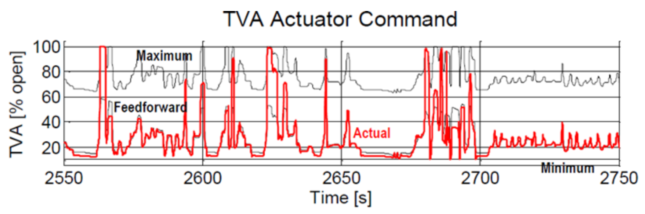


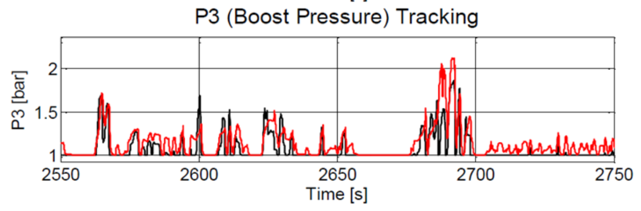
The control problem is expressed here in terms of tracked control inputs (sensor signals) – boost and charge pressure, outputs (actuator commands) – throttle and waste gate and constraints – turbocharger speed and exhaust temperature. The controller is evaluated by applying a given engine speed and boost and charge pressure setpoints and analyzing the results in terms of charging pressure and torque production.

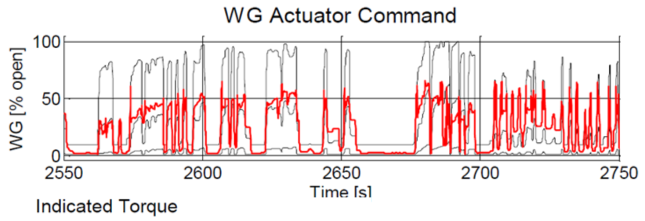
The design of feedback MPC controller consisted of 30 operating points and the controller was designed to heavily prioritize the charge pressure tracking for torque delivery.

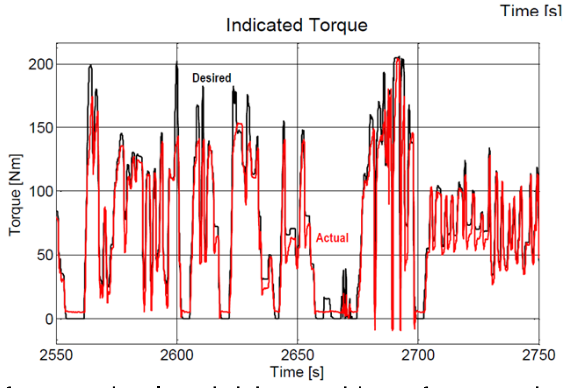
Some typical results generated using the MiL environment including the plant model and the MPC controller are given in the followings. The controller performance is good enough to match the charging pressure requirements using a combined control of the waste gate and throttle, and at the end, insures the right shaft torque production even during the highly transient phases of the ARTEMIS cycle.









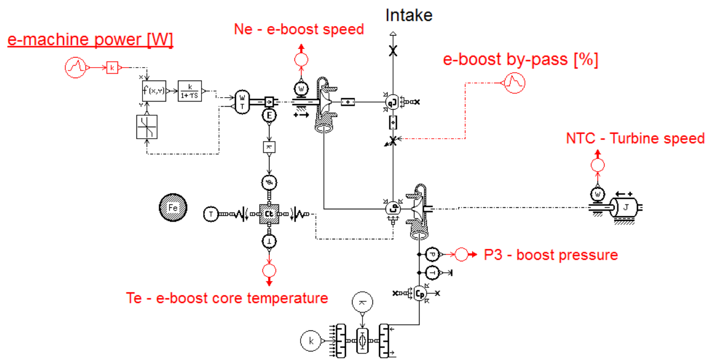


Since the final goal is the introduction of new technology bricks to address future emissions standards (Euro6c and Euro7), from the baseline configuration presented above, RENAULT introduced the electric supercharger technology (e-boost) to evaluate its potential for increasing the engine performance attributes during transient phase of the cycles in particular. Actually, the combined LMS Amesim-OnRAMP approach permitted to rapidly assess several engineering options:

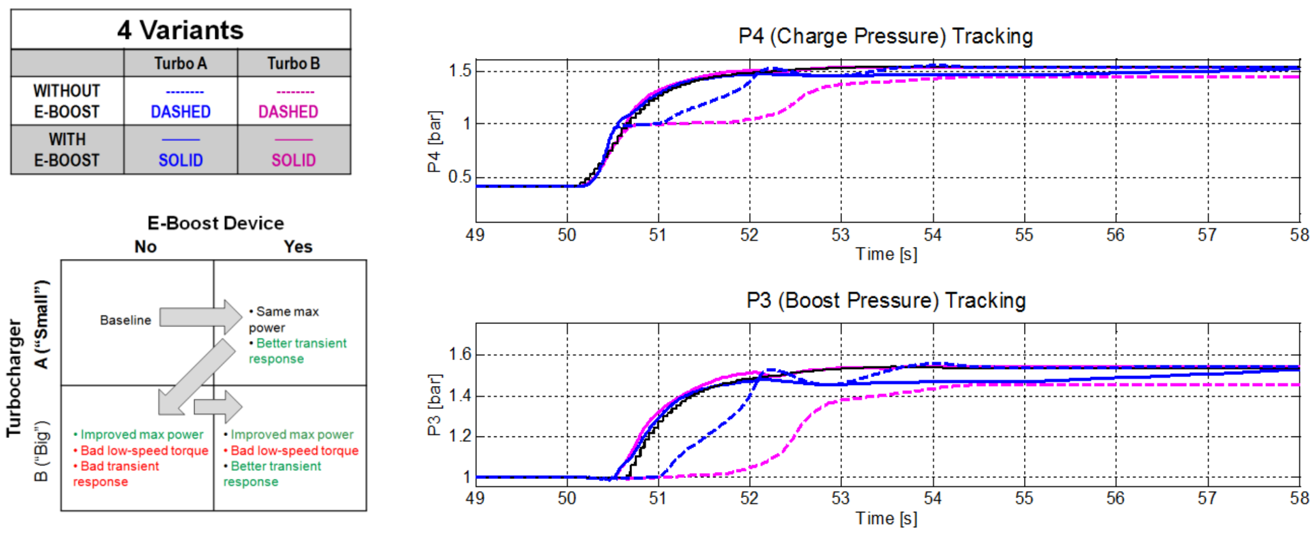
- Architecture: e-boost can be mounted upstream or downstream the main compressor, can be associated to an additional heat exchanger...
- Sizing: power of the e-boost electric machine as a function of the engine performance requirements, e-boost can be combined with an upgraded turbocharger...
- Pollutants: impact on emissions (NOx, HC, CO) on driving cycle and three-way catalytic converter activation time...

Starting from the baseline air path system, the plant model is upgraded including the e-boost machine.

The e-boost device is added to LMS Amesim engine model upstream of the main turbocharger. The control problem is then changed to take into account this additional actuator and new constraints like the thermal state of the electrical machine or its maximum speed.



The figure below shows some results given by the workflow evaluated on four hardware variants – with and without e-boost device and with two version of the turbocharger. This kind of analysis achieved in a couple of hours demonstrates the efficiency and the potential of the tool chain for the selection and pre-sizing of hardware.



Thanks to this approach where MPC is coupled with the plant models, RENAULT is able to complete these numerical studies in a very short time and select the best settings for each project milestones.

6. Conclusion

Two complementary software suites, one for physical modeling and the other for MPC control design, were linked into a new workflow that supports automotive mechanical hardware selection. The workflow allows engineers in charge of advanced engineering and pre-design to investigate in a very short time the potential benefit of a new sub-system--a significant step towards "design-right-the-first-time."

Looking ahead, the same systematic calibration approach and scalability that make MPC attractive for pre-development hardware selection problems, should naturally lead to deployments in production ECUs. RENAULT is currently leading several initiatives in this direction where Renault is evaluating MPC with the use of OnRAMP Design Suite to face scalability and maintainability concerns with production controllers.

Renault is still analyzing how to use MPC on real hardware and industrial application of MPC in powertrain control at Renault remains an open question. Issues yet to be understood better are its impact on ECU metrics and the required changes of the industrial process. Nevertheless, MPC is about to be used for Model based system selection in advanced projects within the company.

References

[1] V.Alfieri, D.Pachner – Enabling Powertrain Variants through Efficient Controls Development, SAE Technical Paper no. 2014-01-1160, SAE World Congress, Detroit, MI, USA, April 8-10 2014.
[2] A.Bemporand, M.Morari, V.Dua, E.N.Pistikopoulos –The Explicit Linear Quadradic Regulator for Constrained Systems, Automatica, 38(1):3-20, 2002.
[3] N.Khaled, M.Cunningham, J.Pekar, A.Fuxman, O.Santin – Multivariable Control of a Dual Loop EGR Diesel Engine with a Variable Geometry Turbo, SAE Technical Paper no. 2014-01-1357, SAE World Congress, Detroit, MI, USA, April 8-10 2014.
[4] Y.Kim, M.Van Nieustadt, G.E.Stewart, J.Pekar – Model Predictive Control of DOC Temperature During DPF Regeneration, SAE Technical Paper no. 2014-01-1165, SAE World Congress, Detroit, MI, USA, April 8-10 2014.
[5] M.Morari –The Role of Theory in Control Practice, Jornadas de Automatica, Plenary Session, Terrassa, Spain, 4-6 September 2013.
[6] D.Pachner, D.Germann, G.E.Stewart – Identification Techniques for Control Oriented Models of Internal Combustion Engines, (Editors: L. Del Re, F. Allgower, L. Glielmo, C. Guardiola, I. Kolmanovsky), p 211-230, Springer 2010.
[7] G.E.Stewart, F.Borrelli, J.Pekar, D.Germann, D.Pachner, D.Kihas – Automotive Model Predictive Control: Models, Methods, and Applications, (Editors: L. Del Re, F. Allgower, L. Glielmo, C. Guardiola, I. Kolmanovsky), p 211-230, Springer 2010.
[8] E.Tseng, D.Hrovat, S.Di Cairano,I.V. Kolmanovsky –The Development of Model Predictive Control in Automotive Industry: A Survey, IEEE MSC, October 3, 2012, Dubrovnik, Croatia.
[9] D.von Wissel, P.Moreno Lahore, *et al*. - Renault Model-Based Design - Powertrain control development process. 23rd Int. AVL Conference “Engine & Environment”, Graz, Austria, Sep.8th - 9th, 2011.
[10] D.von Wissel, A.Husson, V.Talon, L.Lansky, D.Pachner, M.Uchanski - Reducing Engine Calibration Time and Cost with Model Predictive Control, 10th IAV Symposium Automotive Powertrain Control Systems – Berlin, Sep. 11 - 12, 2014
[11] K. Bencherif, D. von Wissel, L. Lansky, D. Kihas - Model Predictive Control as a Solution for Standardized Controller Synthesis and Reduced Development Time Application - Example: Diesel Particulate Filter Temperature Control, SAE 2015 World Congress, Detroit, MI, USA, April 21-23, 2015
[12] V.Talon, V.Thomas - Deployment of system simulation as a support tool for the control development, IFAC ECOSM Congress 2012
[13] Honeywell OnRAMP website: http://www.honeywellonramp.com

Concurrent Programming of Microcontrollers, a Virtual Machine Approach

Steven Varoumas[1,2], Benoît Vaugon[3] and Emmanuel Chailloux[2]

[1]CÉDRIC – Conservatoire national des arts et métiers, Paris, F-75141, France

[2]Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris, France.
steven.varoumas@lip6.fr, emmanuel.chailloux@lip6.fr

[3]U2IS, ENSTA ParisTech, Palaiseau, France
benoit.vaugon@ensta.fr

Abstract

Microcontrollers are low-cost and energy efficient programmable integrated circuits, they are used in a lot of common electronic devices but are quite difficult to program because of very limited resources. Being particularly used for embedded system, they interact a lot with their environment, and should react quickly to external stimuli. In this paper, we study different models of concurrency for programming microcontrollers using a virtual machine approach for safety as well as a higher-level model of programming. We then propose OCaLustre, the prototype of a synchronous extension to OCaml suitable for concurrent programming on microcontrollers.

1 Introduction

Microcontrollers are small integrated circuits that can be considered as simple, albeit complete, computers: they contain a processing core, multiple memory units (typically, nonvolatile memory for program code and volatile memory for data) as well as a set of input/output pins which allow interactions with the surrounding environment of the chip by conducting electric current. Being very small, affordable and energy efficient, microcontrollers are ubiquitous in embedded systems: they can be found implanted in the electronics of common-life objects (such as domestic appliances or toys), as well as in bigger, critical, machines (manufacturing robots, car engines, aircraft systems, . . .), on which they perform tasks of various nature and complexity. Those efficiency advantages come with some drawbacks: microcontrollers typically offer limited processing power and low memory resources, constraining programmers of microcontrollers to use low level programming models in order to keep permanent control of the available hardware resources, memory consumption in particular.

Due to simplicity and performance concerns, programs running on microcontrollers are commonly written in Basic, assembly or subsets of the C language. These programming languages often fail to provide the same level of hardware abstraction, safety, and expressiveness as higher-level programming languages such as Python, Lisp, Java, or OCaml. From this observation, a few virtual machines capable of interpreting the bytecode of such languages have been successfully ported to microcontrollers. These solutions free developers from a lot of hardware considerations, and allow the development of less error-prone and more complex software, providing levels of hardware abstraction, increased safety and a more modern programming style overall.

Typically, embedded systems in which microcontrollers are operating are regularly interacting with the outside world, often reacting to signals sent by various peripherals controlled or not by humans (buttons, sensors, other controllers, etc.). In a lot of cases, all of those different stimuli must be acknowledged and treated as they appear and in any particular order, leading the programs running on microcontrollers to be inherently concurrent. Unfortunately, none of traditional ways of programming microcontrollers (namely imperative low level languages), nor the languages provided by virtual machine approaches are particularly suited for the handling of concurrent tasks for such systems. We thus intend to expand the ways microcontrollers are developed to better comply with the nature of embedded systems, and do so while keeping the safety and expressivity of high-level programming languages as well as a small resources consumption thanks to bytecode factorization and automatic memory management.

In this paper, we study the different ways to improve the programming of microcontrollers with concurrent programming, using a particular virtual machine capable of handling bytecode of the OCaml programming language. We especially focus on providing a high-level model of concurrency adapted to embedded systems and with a low memory consumption in order to comply with the limited resources of microcontrollers. Our different efforts, associated with the development of the OCaPIC virtual machine, lead us to develop OCaLustre, a prototype of a synchronous data-flow extension to OCaml.

2 A virtual machine approach for more powerful programming languages

Resources on microcontrollers, especially memory space, can be very low. For example, the PIC 18 series of microcontrollers commercialized by the *Microchip* company can have at most only 4 kibi-bytes (KiB) of RAM and 128 KiB of program memory. These constraints usually drive the style of programming for such devices to be very low level, with a manual handling of memory usage. For this reason, most microcontrollers are programmed with a subset of C or assembly languages, and programmers need to be knowledgeable about the precise architecture of the chip they use. Not only being quite tedious, not portable, and lacking the richness of constructions of higher-level languages, these approaches also lack important programming safeguards such as static type checking at compile-time or automatic memory management at runtime, leading to the apparition of unforeseen bugs and issues.

In order to free programmers from dealing with tedious tasks relating to the hardware, and to help them develop safer programs, some ports of virtual machines capable of running the bytecode of higher-level programming languages have been completed. These virtual machines, directly implemented in the lower level languages traditionally used for microcontrollers' programming, allow a more portable code and a safer programming model while staying fast and efficient.

Among these various virtual machine approaches, we can mention the Darjeeling Virtual Machine (DVM) [1], a port of the Java virtual machine on Atmel and ARM microcontrollers capable of running a subset of this language. Similarly, the PICBIT [4] and PICOBIT [10] systems allow to run Scheme programs on PIC microcontrollers with virtual stack machines.

Quite paradoxically, these virtual machines approaches can lead to a smaller resulting code (that includes the runtime library and virtual machine) than the corresponding native code because of the more powerful and more complex instruction set of the virtual machines and thanks to bytecode compression and cleaning tools.

OCaPIC: running OCaml bytecode on PIC microcontrollers

The OCaPIC project [11] is a virtual machine approach directed towards running bytecode of the OCaml programming language on the very limited hardware of the PIC 18 microcontrollers. This port of the ZINC Abstract Machine (ZAM) [6] (the original virtual machine of OCaml), written in PIC assembly, allows programmers to use the various advantages of this language and of its runtime on microcontrollers with very scarce resources.

OCaml is a high-level programming language belonging to the ML family of programming languages. Descending from Caml and Caml-light, it was created and maintained at INRIA [1] since 1996. Being multi-paradigm, it implements functional, imperative, modular and object-oriented traits and thus offers a rich expressiveness for writing programs of various nature for embedded systems. Furthermore, OCaml provides a strong static type-checking at compile time, with type inference, which insures the absence of dynamic type error and memory corruption, and thus decreases the amount of possible bugs inside critical applications. Moreover, OCaml comes with a garbage collector (GC) which makes possible an automatic handling of memory resources, and frees programmers from such considerations while providing re-usability of memory. OCaPIC implements two different algorithms of GC: stop-and-copy (by default) and mark-and-compact.

In addition to the port of most of the standard library of OCaml, OCaPIC offers a set of primitives adapted to the handling of the input/output pins: for example, calling the function `set_bit` makes the microcontroller send an electric current on the pin passed in argument and calling `test_bit` reads and returns the state of a given pin.

The following code is an example of a program written with OCaPIC which makes a LED connected to the pin named `RB0` blink every second.

```
open Pic;; (* Module containing write_reg, set_bit, RB0, ... *)
write_reg TRISB 0x00; (* Configure the port B to be an output *)
while true do
    set_bit RB0;
    Sys.sleep 1000;
    clear_bit RB0;
    Sys.sleep 1000;
done
```

After compiling an OCaml program with the standard compiler (`ocamlc`), the resulting bytecode is then cleaned by the provided OCamlClean tool which removes residual dead code. This cleaning operation also has an interesting impact at runtime: the heap is cleaned from some closures and unused global data, typically coming from unused parts of libraries. The resulting bytecode is then compressed and linked with its interpreter into an hexadecimal file that can finally be flashed on a PIC microcontroller. This workflow is depicted in the figure 1.

Finally, OCaPIC comes with two different simulators: the first one interprets the bytecode and makes possible an easy debugging process using usual OCaml tools for correcting errors, such as `ocamldebug`. The second simulator interprets the hexadecimal file produced by OCaPIC and emulates the physical capacities of the microcontroller: it allows checking the native runtime and the different kind of arithmetic and memory overflows. These simulators give a graphical representation of the state of each pin of the microcontroller, in order to check the coherence of

[1] Institut National de Recherche en Informatique et en Automatique

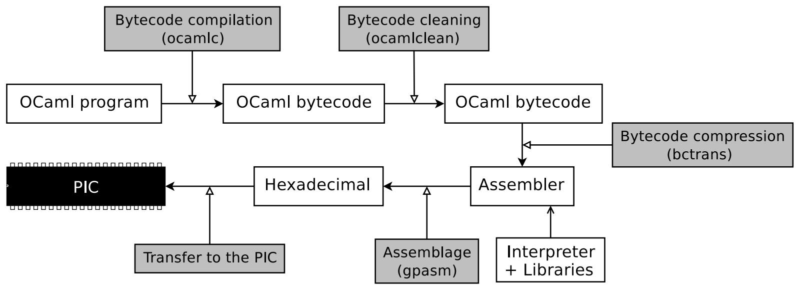


Figure 1: The OCaPIC compilation chain, from OCaml to PIC

the input and output of the programs. They are also able to simulate simple electronic boards connected to the chip (LCD displays, buttons, . . .) for testing purposes.

3 Models of concurrent programming

We engage in improving the virtual machine approach in order to better comply with the nature of embedded programs. Because those systems are in constant interaction with the outside world, for example reacting to button presses, impulses from sensors or signals emitted by other computational devices, we focus on finding a model of concurrent programming adapted to the scarce resources of microcontrollers that could run in the OCaPIC virtual machine.

Analysis of Models of Concurrent Programming

To achieve this goal, we experimented various models of concurrency and analyze their resources consumption and expressiveness.

Firstly, preemptive threading approaches seems to be an ill-adapted model because our microcontrollers do not run any operating system and thus do not provide any underlying scheduling features capable of switching context between tasks depending on priority. Threads directly scheduled by the VM could be conceivable, however they would be quite demanding on memory resources. Moreover, this model of concurrency is not easily predictable and thus any kind of static analysis capable of checking that programs do not go wrong is very difficult to achieve.

Cooperative threading is a far better candidate: letting each process explicitly hand control to other threads does not need the presence of an operating system. We have been successful when porting the core parts of the LWT cooperative thread library [12] into OCaPIC. Reacting to environment stimuli can be done by frequently polling the value on each entry pin of the microcontroller in a separate thread but handling the control points of programs is quite tedious and a single process can block all the system if it does not yield control to the others.

We also managed to port the React functional reactive programming module[2] to OCaPIC, providing a model of concurrency using signals and events appearing and changing through time. While appearing lightweight, this library makes a heavy use of memory allocation, and despite our efforts, we were unable to keep memory use limited to the hardware restrictions.

Lastly, a port of the ReactiveML [7] synchronous language has been performed, but its heavy use of OCaml functors creates a need for too much memory and prevents it for being a viable solution for the concurrent programming of microcontrollers.

[2]See http://erratique.ch/software/react

Nonetheless, the synchronous approach appears to be well-suited for our goal and we decided to direct our work from the control-flow model of ReactiveML to a data-flow synchronous concurrent model.

4 OCaLustre: a synchronous extension to OCaml

All of those experiments have exhibited some major drawbacks for the use of the aforementioned concurrent programming models in real-life applications such as efficiency consideration as well as expressiveness concerns. We thus propose OCaLustre, a synchronous extension to OCaml based on the Lustre [3] dataflow synchronous programming language.

4.1 Syntax and Semantics of OCaLustre

An OCaLustre program is an OCaml program augmented with a construct stemming from Lustre: the node. Nodes are synchronous functions operating on data flows which are considered as being executed instantaneously, following the synchronous hypothesis. The body of each node is equivalent to a system of equations that is solved between two *"ticks"* of the reactive system and these equations, being solved all at once during the same instant, can represent concurrent tasks. We believe that this model is well-adapted to microcontrollers' programming because, at any moment, each of the input/output pin of a chip is either holding an electrical current or not: the absence or presence of current can thus be represented as a boolean data flow and treated by such nodes in order to react "instantaneously" to environment stimuli. The complex algorithmic and computational parts of programs are still handled by traditional OCaml functions that are called inside nodes in order to keep the advantages of the expressiveness of high level constructs.

The syntax of OCaLustre nodes is very close to Lustre, and users familiar with the Lustre style of programming will not really be surprised when using OCaLustre: for example, the code of figure 2 declares a node called `count_pairs` that computes the series of all natural numbers, as well as all the even numbers. The complete syntax for OCaLustre nodes is given on figure 3.

```
let%node count_pairs () (n, m) =
    m := n * 2;
    n := 0 --> pre n + 1
```

Figure 2: An OCaLustre node

All of the OCaml boolean and arithmetic operators can be used inside OCaLustre declarations, as well as the initialization (`->`) and memory (`pre`) operators of Lustre. Note however that the operator `->`, being already reserved by OCaml, has been replaced by `-->` to describe the initialization of data flows. The `when` operator is not usable at this moment, because our OCaLustre prototype doesn't handle nodes running on different clocks (they all run on the default clock). Notice also that type annotations are not requested in OCaLustre, because it is compiled into pure OCaml code that offers a type inference mechanism. In this regard, OCaLustre nodes can handle polymorphic flows, as it is the case in the following code that declares a node testing if the value of a flow `f` has changed between two instants.

```
let%node change (f) (changed) = changed := false --> f <> pre f
```

Adding a new behavior to an existing OCaLustre program is very straightforward: one just have to write new equations providing the new behavior into the involved node(s). Figure 4 presents such a modification: a program waiting for two signals `a` and `b` and returning `o` only

```
       < node > ::=  let%node node_id < flows_sig > < flows_sig >  = < decl_seq >
  < flows_sig > ::= (id[,id] *) | ()
   < decl_seq > ::= < declaration > | < decl_seq >;< declaration >
< declaration > ::= < flow > := < expression >
       < flow > ::= id | (id, id)
   < infix_op > ::= + | - | * | / |+. | -. | *. | /. | < | > | <= | >= | = | <> | && | ||
  < prefix_op > ::= not
   < constant > ::=  int | bool | float | ()
 < parameters > ::= (< parameter >[,< parameter >] *) | ()
  < parameter > ::= id | < constant >
 < expression > ::= < constant > | id
                  | if < expression > then < expression > else < expression >
                  | < expression > < infix_op > < expression >
                  | < expression > --> < expression >
                  | < prefix_op > < expression > | pre < expression >
                  | call ocaml_function_id < parameters >
                  | node_id < parameters > | (< expression >,< expression >)
```

Figure 3: The syntax of OCaLustre

when r is not present is easily expanded into a program waiting for the presence of a third signal c. This process doesn't lead to any explosion of memory usage when the resulting program is executed, due to the lightweight model of compilation of Lustre.

```
let%node edge (x) (y) =
   y = false --> x && not pre x

let%node abro (a,b,r)  (o) =
   o := edge (seenA && seenB);
   seenA := false -->
      not r && (a || pre seenA);
   seenB := false -->
      not r && (b || pre seenB)
```

```
let%node abcro (a, b, c, r) (o) =
   o := edge (seenA && seenB &&
              seenC);
   seenA := false -->
      not r && (a || pre seenA);
   seenB := false -->
      not r && (a || pre seenB);
   seenC := false -->
      not r && (c || pre seenC)
```

Figure 4: Adding a new behavior to an OCaLustre node

4.2 Compilation of OCaLustre

An OCaLustre program is compiled following the Lustre model of "simple-loop" compilation: each node is converted into a sequential function, and the main node of the program is run inside a single endless loop. Following this method, an OCaLustre program is compiled into a pure OCaml program that can then be handled by OCaPIC as any other OCaml program (see figure 5). Note that, because these two steps are entirely independent, the OCaml code produced after compilation of OCaLustre can also be used with any other OCaml interpreter or compiler: this makes possible to use our extension with any kind of hardware target supported by OCaml.

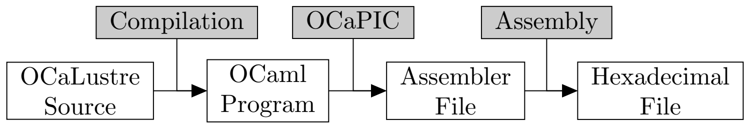


Figure 5: From OCaLustre to executable through pure sequential OCaml

During compilation, each node is replaced by an OCaml function by converting all of the declarations contained inside its body into a sequence of assignations. In OCaLustre, the equation order does not matter (we may use a flow before declaring it) but in OCaml this order matters and a reordering of each declaration is completed at compile-time. This reordering corresponds to a rescheduling of each task of our concurrent program. This process may fail when two flows depend on each other at the same instant: this kind of behavior is invalid since it denotes a causality loop inside the program and an error is then raised by the compiler.

Each usage of the memory operator "`pre`" is converted into an OCaml reference holding an option type. At the first instant, all references hold the value `None`, and at instant $i + 1$, they contain the value of its flow at instant i.

The --> initialization operator is converted into a simple conditional operator checking with a simple boolean value `init` if the function is executed at the first instant or not.

In order to preserve the execution context of a node, a closure is returned by each instantiation function corresponding to a synchronous node. This closure is bound to all of the registers of the node. All of these compiling rules are depicted in the example of figure 6 which illustrates code generation for the node `count_pair` of figure 2.

```
let count_pairs () =
   let init = ref true in
   let pre_n = ref None in
   let count_pairs_step () =
      let n = if !init then 0 else Option.get (!pre_n) + 1 in
      let m = n * 2 in
      init := false; pre_n := Some n; (n, m) in
   count_pairs_step
```

Figure 6: Compilation of an OCaLustre node

4.3 Benchmark analysis

The compilation model of OCaLustre is quite efficient and makes the resulting compiled programs very small. In order to compare the efficiency of OCaLustre with the various models of concurrency that we had firstly considered, we developed a simple application in which two different counters are concurrently displayed on an LCD screen. Four versions of this application have been developed with OCaLustre as well as with three other models: cooperative threading with LWT, functional reactive programming with React and a simple sequential program written in pure OCaml. The size of the programs generated by OCaPIC after a cleaning pass by OCamlClean (containing the runtime library, the virtual machine as well as the compressed bytecode) allows us to assert that OCaLustre is a very lightweight solution, using about 1.5 times less space than LWT and being more than three times smaller than the React solution. In fact, the OCaLustre application is very close, in size, to the compiled sequential code. We executed the programs on our microcontroller (a PIC 18F4620) and analyzed the dynamic memory allocation of each tool when the program is loaded. We found again that OCaLustre is very low demanding in resources with React and LWT allocating respectfully 1668 and 1136 bytes while OCaLustre only using 272 bytes of dynamic memory space:

Tool	React	LWT	OCaLustre	Sequential Code
Size of the program	23.8 KiB	11.7 KiB	**7.8 KiB**	6.8 KiB
Initial dynamic allocation	1668 B	1136 B	**272 B**	150 B

Using our OCaLustre prototype, we developed a program for a device capable of tempering chocolate. This device receives inputs from a temperature sensor and from two buttons (labeled "+" and "-") used to set a desired temperature for the chocolate. Its outputs are an LCD displaying the desired temperature, as well as the current temperature of the chocolate, and a set of resistances capable of heating the preparation. The following OCaLustre program controls the tempering machine by computing a value (`prop`) that represents the amount of time at which the resistances need to be on, depending on the difference between the desired temperature (`wtemp`) and the actual temperature (`ctemp`). The `main` node of the program receives at each instant the value of the buttons as well as the current temperature, computes the value of the desired temperature, the state of the device (on or off) and decides if the heating resistances must be active based on all of these informations:

```
(* Temperature in celsius is (1033-ctemp)/11.67 *)
let%node update_prop (wtemp, ctemp) (prop) =
    new_prop := 0 --> (min (100, max (0, pre new_prop + offset)));
    delta    := min (10, max (-10, ctemp - wtemp));
    delta2   := if delta < 0 then -delta * delta else delta * delta;
    offset   := min (10, delta2);
    prop     := new_prop / 10

let%node timer (number) (alarm) =
    time  := 1 --> if pre time = 100 then 1 else pre time + 1;
    alarm := time < number * 10

let%node heat (w, c) (h) =
    prop := update_prop (w, c);
    h    := timer (prop)

let%node change_wtemp (state) (w) =
    w := 654 --> if state = 1 then pre w - 1 else
                 if state = 2 then pre w + 1 else pre w

let%node thermo_on (state) (on) =
    on := true --> if state = 3 then not (pre on) else pre on

let%node main (plus, minus, ctemp) (wtemp, on, heat) =
    state := call (buttons_state plus minus);
    on    := thermo_on (state);
    wtemp := if on then change_wtemp (state) else 0;
    heat  := if on then heat (wtemp, ctemp) else false;
```

An application displaying the same behavior was also developed in classic OCaml and used in OCaPIC before the development of OCaLustre, so we are able to compare the size of our OCaLustre chocolate tempering program versus the size of the OCaml one, as well as display the significant size difference between the generated OCaml bytecode and the resulting compressed OCaPIC program:

Language	OCaml		OCaLustre	
Type	Bytecode File	PIC Program	Bytecode File	PIC Program
Size	268 KiB	27 KiB	258 KiB	**27 KiB**

We conclude from these results that, after cleaning and compression, the PIC program is a lot smaller than the original bytecode file. Judging by various experiments, this decrease by a factor of about 10 is not uncommon. We show once again that the OCaLustre mode of compilation keeps programs very lightweight and thus allow to run real world programs on devices with small resources.

5 Conclusion

Our efforts dedicated at proposing a higher level of programming for microcontrollers seem promising. In particular, OCaLustre, a synchronous extension to the multiparadigm OCaml programming language, makes possible an easier development of concurrent programs aimed at embedded systems. This extension, based on the data flow language Lustre is quite similar to the Lucid Synchrone programming language [2] but our prototype offers a simpler model. It is translated into pure OCaml and compiled into bytecode that can be lightened by OCamlClean. This model makes the program written in OCaLustre smaller than the ones developed in the richer Lucid language which unfortunately (and contrary to OCaLustre[3]) is not distributed with an open source license, which was necessary for our early experiments with models of concurrency.

Real-time considerations for programs developed with OCaLustre are usual: Worst Case Execution Time (WCET) of the synchronous parts of OCaLustre programs can be computed using classical tools for synchronous programming. For the algorithmic part written in OCaml, we need a guarantee that there is enough memory and to consider dynamic memory management (garbage collector or GC) execution time for WCET analysis. These two notions are nested but depend on the programming style of the algorithmic part and need to deal with dynamic memory allocations (stack and heap) for recursive functions and dynamic data structures. One possibility will be to use a real-time GC [5] but they have important costs (time and memory) and not really adequate for the low resources of PIC micro-controllers. For that, we need to help the GC by giving indications on liveness of objects. Some on-going works can use static region analysis for a mini-ML. In this case freeing a region can be immediate or can allow to manually trigger the garbage collector. The virtual machine approach is not detrimental to the real-time aspects, actually it also makes possible the factorization of the WCET by measuring it directly on the bytecode and then extrapolate it for different hardware.

Moreover, the use of a virtual machine facilitates the debugging and tracing of programs: without modifying program sources the interpreter can be instrumented in order to offer useful informations, allowing a non-intrusive structural code coverage like Zamcov project [13] for the Ocaml Virtual Machine (ZAM). The association of functional languages with critical embedded systems was not previously unseen and have already been used (with less hardware constraints) for the development of various tools, in particular of code generators (like KCG, the code generator of the SCADE SUITE™ [8]).

Using this virtual machine approach, we are able to produce lightweight and portable code which offers higher-level guarantees thanks to the use of the OCaml programming language. The different tools of code analysis are also factorized by the use of bytecode and can be adapted for many different microcontrollers, offering more safety for embedded system applications. Finally, our synchronous extension of OCaml offers a deterministic model of concurrence adapted to the nature of embedded system and the analysis of the safety of programs and its association with an OCaml virtual machine makes possible the development of richer and safer applications.

We aim for future works at improving the OCaLustre language and developing more serious applications (for example in robotics or home automation) while trying to offer formal guaran-

[3]`https://github.com/stevenvar/OCaLustre/`

tees for the execution of programs inside devices fitted with minimal memory and computing resources. As for OCaPIC, a couple of similar projects aimed at porting the virtual machine of OCaml for Atmel AVR and STM32 microcontrollers (respectively used in Arduino and Nucleo platforms) have been instantiated.

References

[1] Brouwers, N., Corke, P., and Langendoen, K. Darjeeling, a Java Compatible Virtual Machine for Wireless Sensor Networks. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion* (2008).

[2] Caspi, P., Hamon, G., and Pouzet, M. *Real-Time Systems: Models and verification — Theory and tools.* ISTE, 2007, ch. Synchronous Functional Programming with Lucid Synchrone.

[3] Caspi, P., Pilaud, D., Halbwachs, N., and Plaice, J. A. Lustre: A declarative language for real-time programming. In *In 14th Symposium on Principles of Programming Languages (POPL'87). ACM* (New York, NY, USA, 1987), POPL '87, ACM, pp. 178–188.

[4] Feeley, M., and Dubé, D. Picbit: a Scheme system for the PIC microcontroller. In *Scheme and Functional Programming Workshop (SFPW'03)* (Nov. 2003), pp. 7–15.

[5] Jones, R., Hosking, A., and Moss, E. *The Garbage Collection Handbook: The Art of Automatic Memory Management.* CRC Applied Algorithms and Data Structures. Chapman & Hall, Jan. 2012.

[6] Leroy, X. The ZINC experiment : an economical implementation of the ML language. Tech. Rep. RT-0117, INRIA, Feb. 1990.

[7] Mandel, L., and Pouzet, M. ReactiveML, a reactive extension to ML. In *Proceedings of 7th International conference on Principles and Practice of Declarative Programming (PPDP 2005)* (Lisbon, Portugal, July 2005).

[8] Pagano, B., Andrieu, O., Moniot, T., Canou, B., Chailloux, E., Wang, P., Manoury, P., and Colaço, J.-L. Experience Report: Using Objective Caml to develop safety-critical embedded tool in a certification framework. In *International Conference of Functional Programming ICFP 09* (2009), pp. 215–220.

[9] St-Amour, V., and Feeley, M. Picobit: A Compact Scheme System for Microcontrollers. In *International Symposium on Implementation and Application of Functional Languages (IFL'09)* (Sept. 2009), pp. 1–11.

[10] Vaugon, B., Wang, P., and Chailloux, E. Programming Microcontrollers in Ocaml: the OCaPIC Project. In *International Symposium on Practical Aspects of Declarative Languages (PADL 2015)* (June 2015), no. 9131 in Lecture Notes in Computer Science, Springer Verlag, pp. 132–148.

[11] Vouillon, J. Lwt: A cooperative thread library. In *Proceedings of the 2008 ACM SIGPLAN Workshop on ML* (2008), ML '08, ACM, pp. 3–12.

[12] Wang, P., Jonquet, A., and Chailloux, E. Non-Intrusive Structural Coverage for Objective Caml. In *5th Workshop on Bytecode Semantics, Verification, Analysis and Transformation* (2011), vol. 264 4 Electronic Notes in Theoretical Computer Science, Elsevier, pp. 59–73.

Session 28

Method and tools

Friday 29th, 09:00 – Ariane 2

Asynchronous modeling in railway systems

Emmanuel Gaudin
PragmaDev, France
emmanuel.gaudin@pragmadev.com

Abstract: *Models in the railway industry are often based on synchronous technologies such as Matlab or Scade. This is due to technical reasons, but because of its concepts the abstraction level of synchronous models are very low and very close to the implementation level. A serious gap is observed between the requirements described in natural textual language and the model which is basically an implementation. The increasing level of system complexity, combining communicating subsystems, calls for a more abstract model. This paper will first discuss why synchronous technologies have been used in this type of systems, then an experiment of using an asynchronous technologies on a real ERTMS case coming from SNCF is described, and finally the paper will conclude on how an asynchronous modeling technologies could make the link between the informal textual requirements and the implementation of the system.*

Keywords*: Modeling, Asynchronous, Synchronous, Matlab, Lustre, SDL, TTCN-3, Railways, ERTMS*

Introduction

When it comes to modeling two main questions have to be addressed. The first one is about positioning the model in the development cycle ; defining if the model is a requirement, a specification, or a design. The second one is about the modeling technology to use ; depending on what the model is aiming at. The lower is the model level, the more specialized is the modeling technology, and the narrower is the scope of the model.
In [1] the technologies usually applied to model train systems are listed such as the B method, Scade, Simulink/Stateflow. In [2] and [3] the authors present how they have written a specific type of model in order to verify specific safety properties. The models are usually dedicated to the targeted model checking technology and can not be used for anything else.
In [4] is presented the work done by SNCF to verify safety rules using The Mathworks tools.
In [5] the author presents a tool that makes a link between a system level model written with Papyrus SysML modeler and a design level model written with SCADE Suite.
In [6], following the ASSERT FP6 european project, the European Space Agency has been promoting the TASTE (The ASSERT Set of Tools for Engineering) framework. Because each technology is best suited for a part of the overall system, TASTE framework aims at gathering the different technologies in a consistent framework. The top level model is an architecture model based on AADL and ASN.1. The different AADL architecture blocks are further developed with a dedicated technology such as Scade or SDL. When all models are validated a code generator automatically gathers the code generated by the different tools.

In the above references, the choice of the modeling language is often driven by the possible verification associated to the technology. For that purpose models are based on low level modeling technologies that are very close to the implementation details.
Attempts to raise up the modeling level have been done using a combination of languages. For that purpose the synchronous or asynchronous approaches are put at the same level and a third language is used as an overall model view (SysML or AADL). In this paper we are experimenting a different approach in which an asynchronous SDL model is used as a bridge between the requirements and a low level synchronous model. To demonstrate this, an existing Matlab model is taken as an example and translated to an SDL model. Using an SDL simulator and solver the system functions are then analyzed. Finally there will be a discussion on what the SDL model brings to a Matlab model.

A natural synchronous approach

In the old days, train systems were exchanging simple information such as “is the train present on that portion of track” , “are the doors open or not”. These information can be detected with simple electrical detectors along the tracks, on the platforms, or in the train itself. These detectors behave like electrical switches and the information can be extracted from the fact that the circuit is open or closed. From a software point of view, all this information can be represented by binary variables. The rationale is a logical combination of the different information gathered. It would be something like “if the train is not stopped at the platform, then the doors should be closed”. And this information would be verified every time some new information was received from the sensors. Each clock tick, the information from the sensors is gathered and given to a logical system that will compute all the entries and produce some outputs.
That is the basic principle of a synchronous approach. All the entries are valid at the same time, some logical operators combine the inputs taking into consideration the previous values of the inputs, and produces some outputs. Each clock tick, the whole information is computed again and again. The fact that the system re-computes everything at every clock tick actually reduces the possible number of cases that might occur in the system. It is therefore much easier to verify properties at each clock tick and make sure the system behaves properly whatever happens, whatever the information read by the sensors.

New approach for upcoming systems

Train management systems are evolving, and in particular the European Rail Traffic Management System (ERTMS [7]), which was initiated in the 90’s by the European Union, aims at harmonizing and expanding the capabilities of train control systems over Europe so that a train crossing borders does not need a specific controller for each country it crosses. This standard covers specification of on board equipments, on the trackside equipments, as well as communication information systems. The information exchanged includes speed, acceleration and so on. It is more and more complex and is getting quite far from the original binary information. Furthermore all the equipments are not mechanically or electrically connected, they are now completely desynchronized from one another. The information is not that simple any more, it is complex and unpredictable. Using synchronous technologies might work on a local level but will definitely not be sufficient to describe new features that combine a lot of communication. In fact the higher is the level of the view, the less a synchronous description will fit. This is a known issue in systems where complexity is increasing. It has been theorized and discussed in several publication as the GALS (Globally Asynchronous Locally Synchronous [10]) theory of system description since the 80s. At that time, asynchronous descriptions were transformed to synchronous descriptions based on the theory that asynchronous models could be deconstructed to synchronous models and reconstructed back [11]. The point was to be able to use the existing and mature synchronous validation and verification technologies on the market at that time [12]. This works as long as the model is close to the implementation. This is not satisfying any more in complex communicating systems as the first thing to do when developing a new feature in a system is to verify its functionality before trying to implement it. That raises the need for asynchronous descriptions and verification techniques.

An asynchronous description of existing models

On one side the higher the abstraction level is, the more asynchronous are the relations among the elements in the system. On the other side in order to produce a relevant and verifiable dynamic description, the model needs to be executable. That means it should be statically and dynamically unambiguous from a semantic point of view.

SDL [6] international standard that was initially designed to describe telecommunication protocols is a good candidate for this type of description. It is by nature asynchronous, it combines graphical views for architecture and state machines, and includes an action language with simple data types to describe a detailed behavior whenever needed in the description.

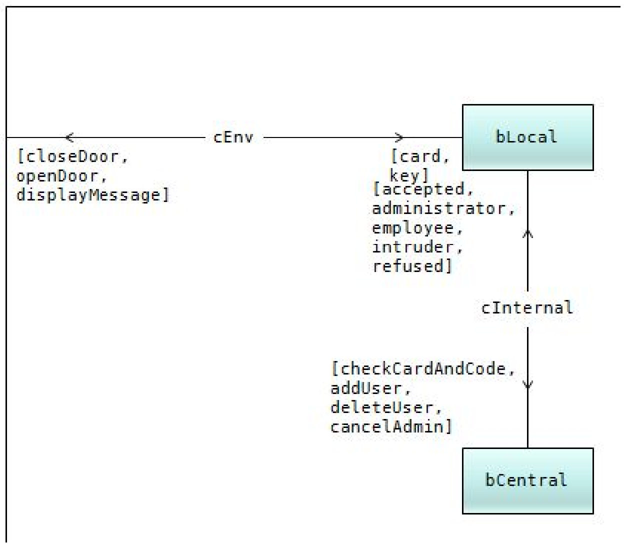


Figure 1: A basic SDL architecture

Figure 1 shows an SDL architecture with two blocks. Each block can be further decomposed in sub-blocks. At the lower level of the architecture one or several finite state machines describe the behavior of its container. The flow of information between the blocks is message based. Each state machine has its own implicit FIFO message queue.There is no clock based inputs in an SDL system. Only the sequence of events matters.

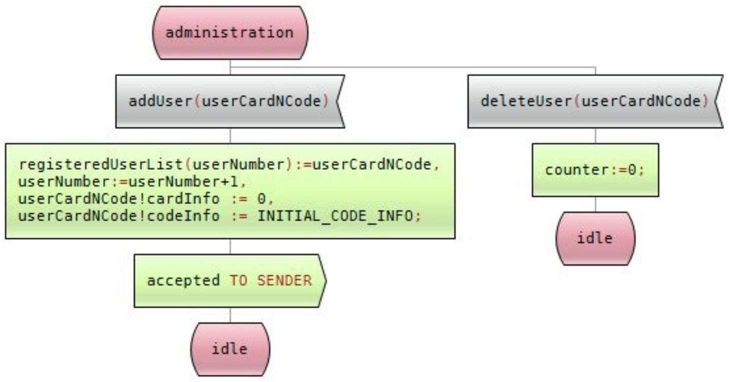

administration
addUser(userCardNCode)
deleteUser(userCardNCode)
registeredUserList(userNumber):=userCardNCode,
userNumber:=userNumber+1,
userCardNCode!cardInfo := 0,
userCardNCode!codeInfo := INITIAL_CODE_INFO;
counter:=0;
idle
accepted TO SENDER
idle

Figure 2: A simple SDL state machine

Figure 2 shows a simple SDL state machine. In the *administration* state, the state machine will read its message queue. If *addUser* message is received the instructions below the *addUser* input symbol are executed, an *accepted* message is sent to the sender of the *addUser* message, and the transition ends back in the *idle* state. If message *deleteUser* is received, the counter internal variable is set to 0, and the transition ends also in the *idle* state.
The different state machines in an SDL model run in parallel. The main issue with this type of description is verification. Because of its asynchronous nature, events can occur at any time, independently from each other, and this creates a huge number of possible scenarios. Model checking tools can explore the possible combinations, but the number of cases is sometimes very difficult to handle making verification of properties on this type of system a real challenge.
Since this type of description is well suited for a high level description it is naturally close to a functional description or a high level requirement. It is therefore quite interesting to analyze how the requirements could be translated into an asynchronous model and a synchronous model, and see if one could be translated to the other. This is what was done on a Radio Block Center from the ERTMS[7]. Figure 3 and 2 show the architecture level using Matlab and using SDL. Even though both model contain the same blocks the main difference is the communication semantic. Information exchange is synchronous in Matlab and message based in SDL.

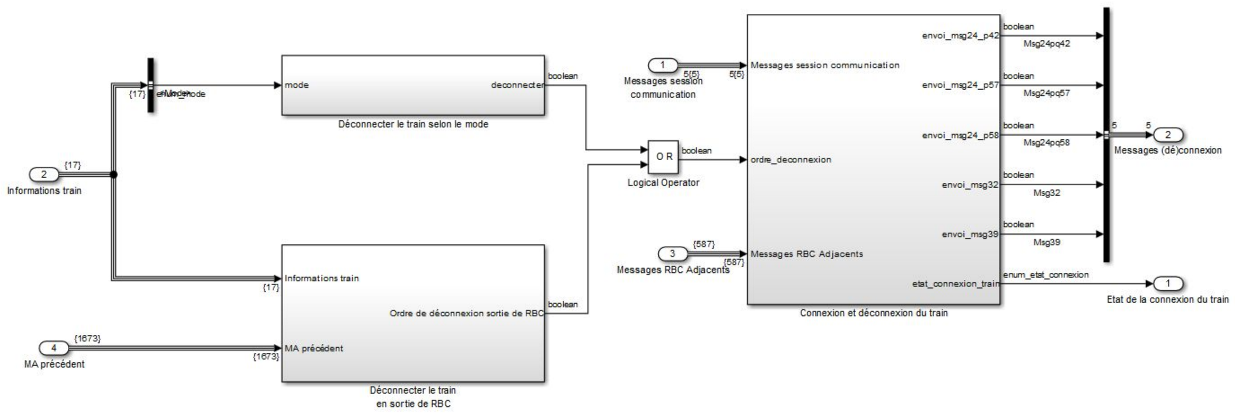


Figure 3: Architecture described with Matlab

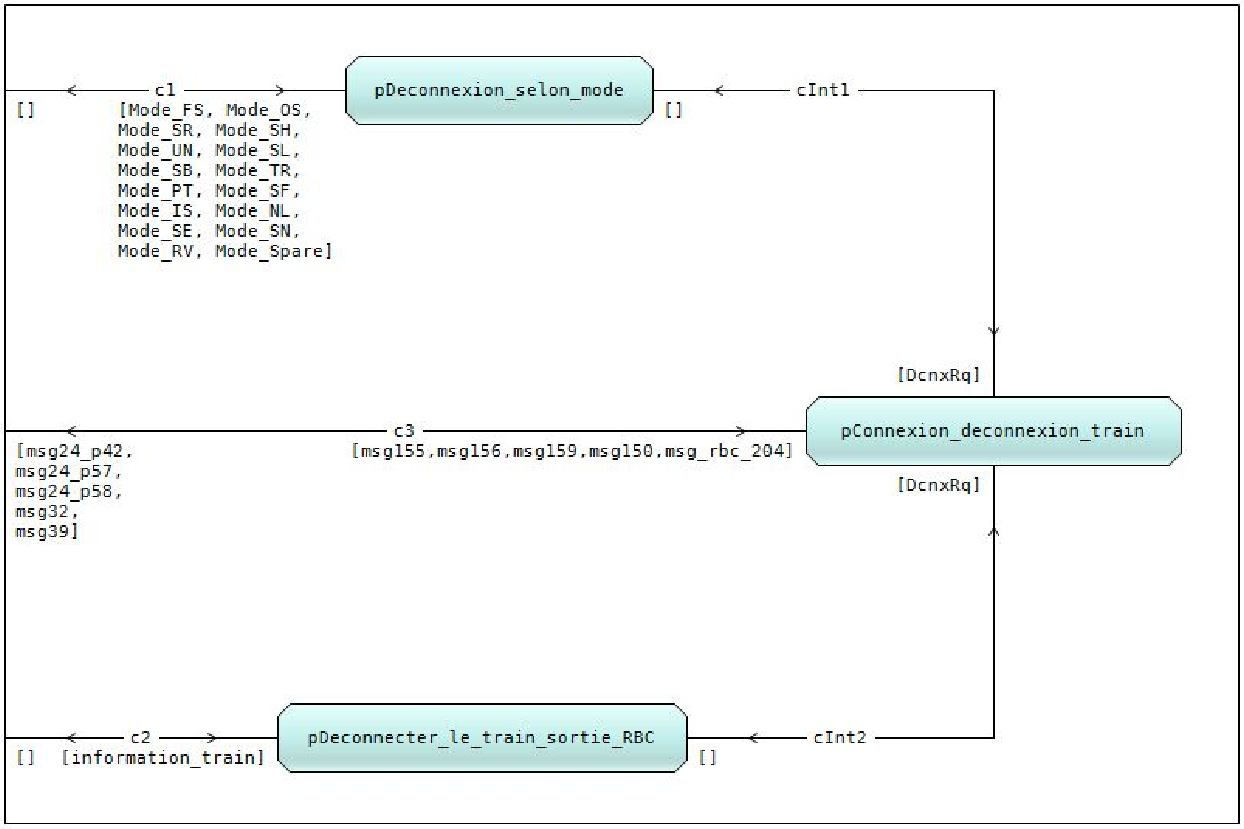

c1
[]
[Mode_FS, Mode_OS, Mode_SR, Mode_SH, Mode_UN, Mode_SL, Mode_SB, Mode_TR, Mode_PT, Mode_SF, Mode_IS, Mode_NL, Mode_SE, Mode_SN, Mode_RV, Mode_Spare]
pDeconnexion_selon_mode
[]
cInt1
[DcnxRq]
c3
[msg24_p42, msg24_p57, msg24_p58, msg32, msg39]
[msg155,msg156,msg159,msg150,msg_rbc_204]
pConnexion_deconnexion_train
[DcnxRq]
c2
[]
[information_train]
pDeconnecter_le_train_sortie_RBC
[]
cInt2

Figure 4 - Architecture in SDL with three state machines

Matlab diagram in Figure 5 indicates the description of the block is done with a state machine. It lists all the inputs and outputs of the state machine. This is not necessary in SDL as a process behavior is always described with a state machine.

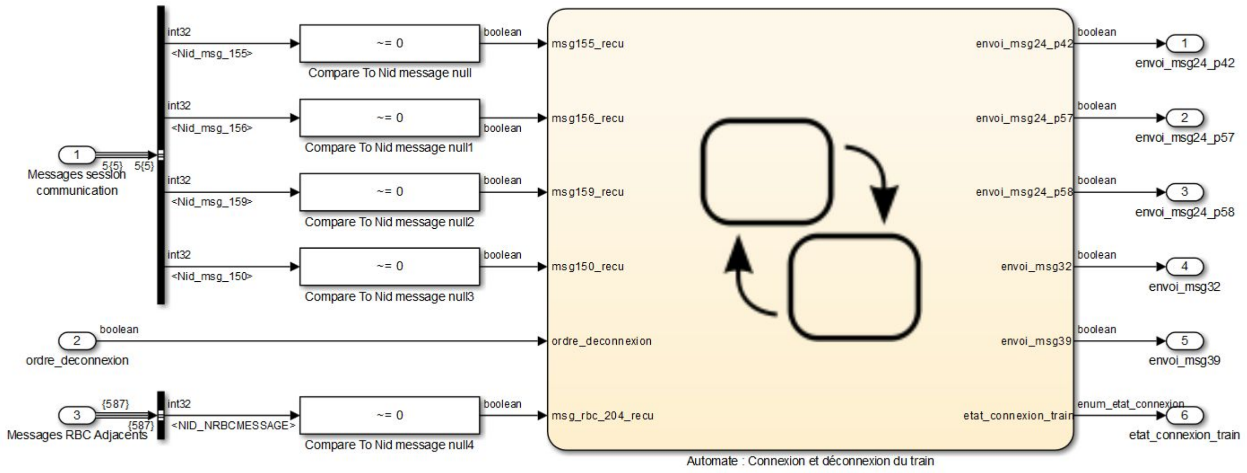
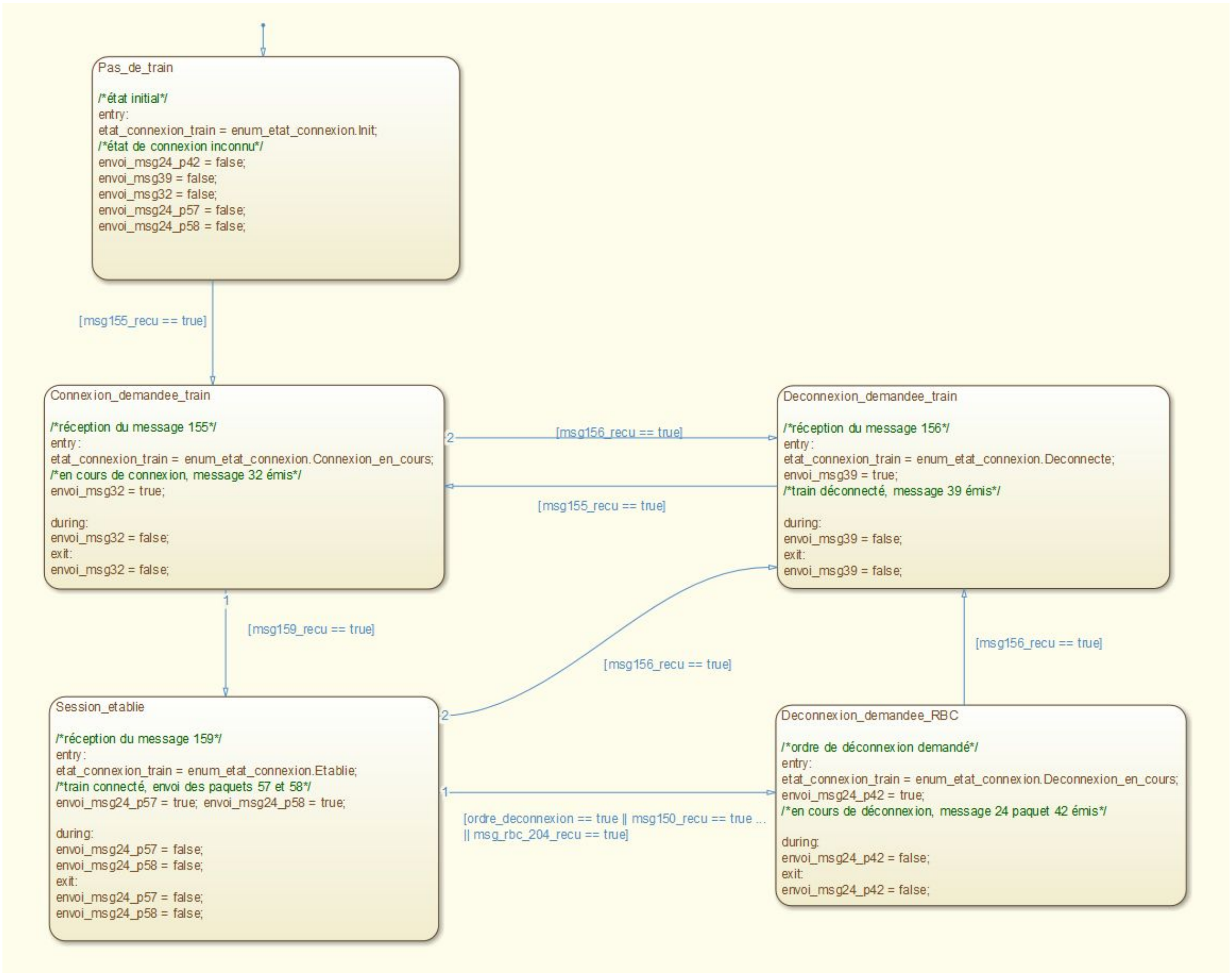


Figure 5 : Connect and disconnect block is made of one synchronous state machine

The Matlab state machine is described in Figure 6 and the SDL equivalent state machine is described in Figure 7.


```
Pas_de_train

/*état initial*/
entry:
etat_connexion_train = enum_etat_connexion.Init;
/*état de connexion inconnu*/
envoi_msg24_p42 = false;
envoi_msg39 = false;
envoi_msg32 = false;
envoi_msg24_p57 = false;
envoi_msg24_p58 = false;
```

```
Connexion_demandee_train

/*réception du message 155*/
entry:
etat_connexion_train = enum_etat_connexion.Connexion_en_cours;
/*en cours de connexion, message 32 émis*/
envoi_msg32 = true;

during:
envoi_msg32 = false;
exit:
envoi_msg32 = false;
```

```
Deconnexion_demandee_train

/*réception du message 156*/
entry:
etat_connexion_train = enum_etat_connexion.Deconnecte;
envoi_msg39 = true;
/*train déconnecté, message 39 émis*/

during:
envoi_msg39 = false;
exit:
envoi_msg39 = false;
```

```
Session_etablie

/*réception du message 159*/
entry:
etat_connexion_train = enum_etat_connexion.Etablie;
/*train connecté, envoi des paquets 57 et 58*/
envoi_msg24_p57 = true; envoi_msg24_p58 = true;

during:
envoi_msg24_p57 = false;
envoi_msg24_p58 = false;
exit:
envoi_msg24_p57 = false;
envoi_msg24_p58 = false;
```

```
Deconnexion_demandee_RBC

/*ordre de déconnexion demandé*/
entry:
etat_connexion_train = enum_etat_connexion.Deconnexion_en_cours;
envoi_msg24_p42 = true;
/*en cours de déconnexion, message 24 paquet 42 émis*/

during:
envoi_msg24_p42 = false;
exit:
envoi_msg24_p42 = false;
```

Figure 6 : Matlab synchronous state machine

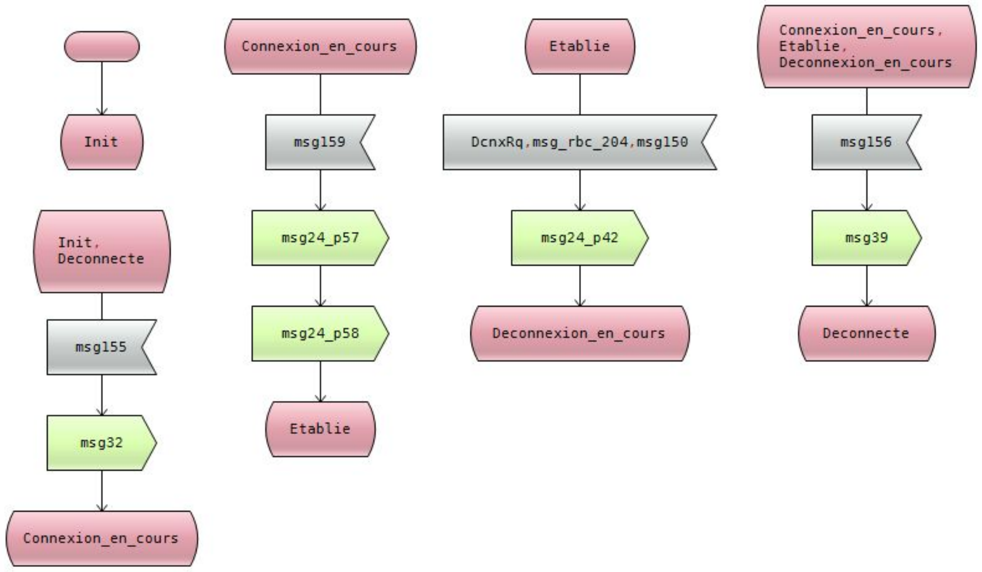
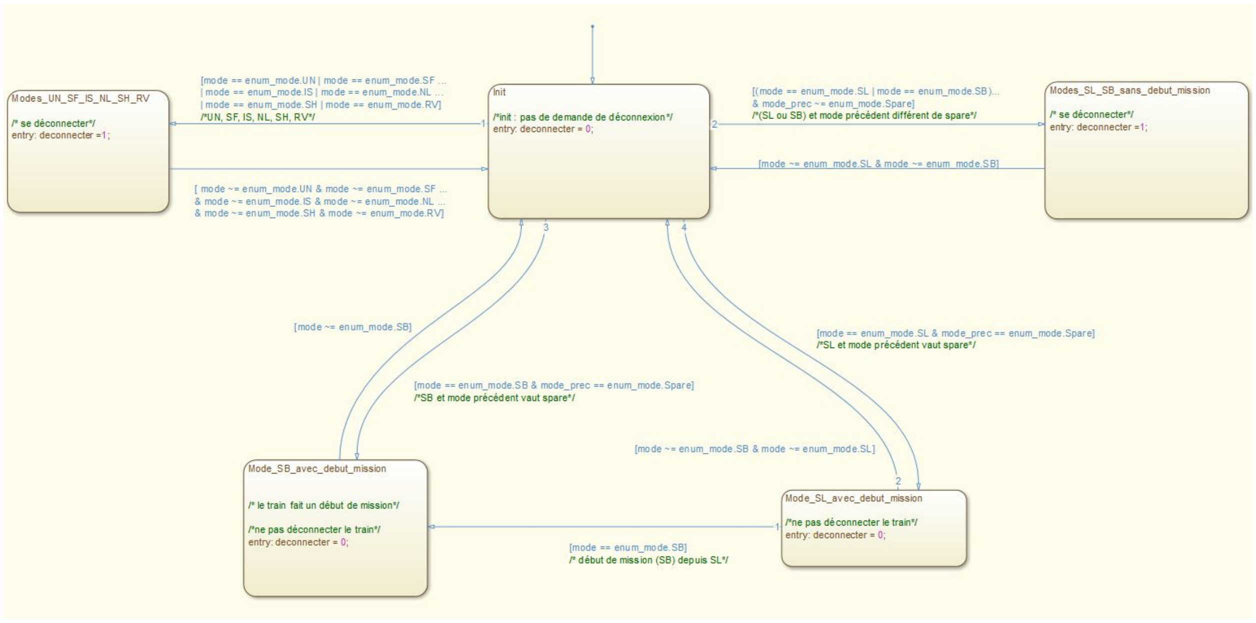


Figure 7 : SDL asynchronous state machine

In the example described in Figure 6 & 7 the inputs are very similar. For example reception of msg155 event is done setting the boolean variable msg155_recu to true in Matlab, while it uses the message input symbol in SDL. The Matlab representation forces the modeler to make sure msg155_recu is set to false after being received because if not it might be taken into consideration again in another transition. Similarly to output some information from the state machine, the Matlab model sets boolean values to true or false. For example in the Session_etablie state, envoi_msg32 is set to true when entering the state, then set to false while in the state, and again set to false when exiting the state. In an event based language such as SDL, there is only one msg32 output when msg155 is received and that's it. In that sense it makes things much clearer.

The other example below, Deconnecter_selon_mode in Figure 8 & 9, shows how to disconnect the train depending on the mode in which it is.



Modes_UN_SF_IS_NL_SH_RV
/* se déconnecter*/
entry: deconnecter =1;
[mode == enum_mode.UN | mode == enum_mode.SF ...
| mode == enum_mode.IS | mode == enum_mode.NL ...
| mode == enum_mode.SH | mode == enum_mode.RV]
/*UN, SF, IS, NL, SH, RV*/
1
[mode ~= enum_mode.UN & mode ~= enum_mode.SF ...
& mode ~= enum_mode.IS & mode ~= enum_mode.NL ...
& mode ~= enum_mode.SH & mode ~= enum_mode.RV]
Init
/*init : pas de demande de déconnexion*/
entry: deconnecter = 0;
2
[(mode == enum_mode.SL | mode == enum_mode.SB)...
& mode_prec ~= enum_mode.Spare]
/*(SL ou SB) et mode précédent différent de spare*/
Modes_SL_SB_sans_debut_mission
/* se déconnecter*/
entry: deconnecter =1;
[mode ~= enum_mode.SL & mode ~= enum_mode.SB]
3
4
[mode ~= enum_mode.SB]
[mode == enum_mode.SB & mode_prec == enum_mode.Spare]
/*SB et mode précédent vaut spare*/
[mode == enum_mode.SL & mode_prec == enum_mode.Spare]
/*SL et mode précédent vaut spare*/
[mode ~= enum_mode.SB & mode ~= enum_mode.SL]
2
Mode_SB_avec_debut_mission
/* le train fait un début de mission*/
/*ne pas déconnecter le train*/
entry: deconnecter = 0;
1
[mode == enum_mode.SB]
/* début de mission (SB) depuis SL*/
Mode_SL_avec_debut_mission
/*ne pas déconnecter le train*/
entry: deconnecter = 0;

Figure 8 - "Disconnect depending on the mode" Matlab state machine

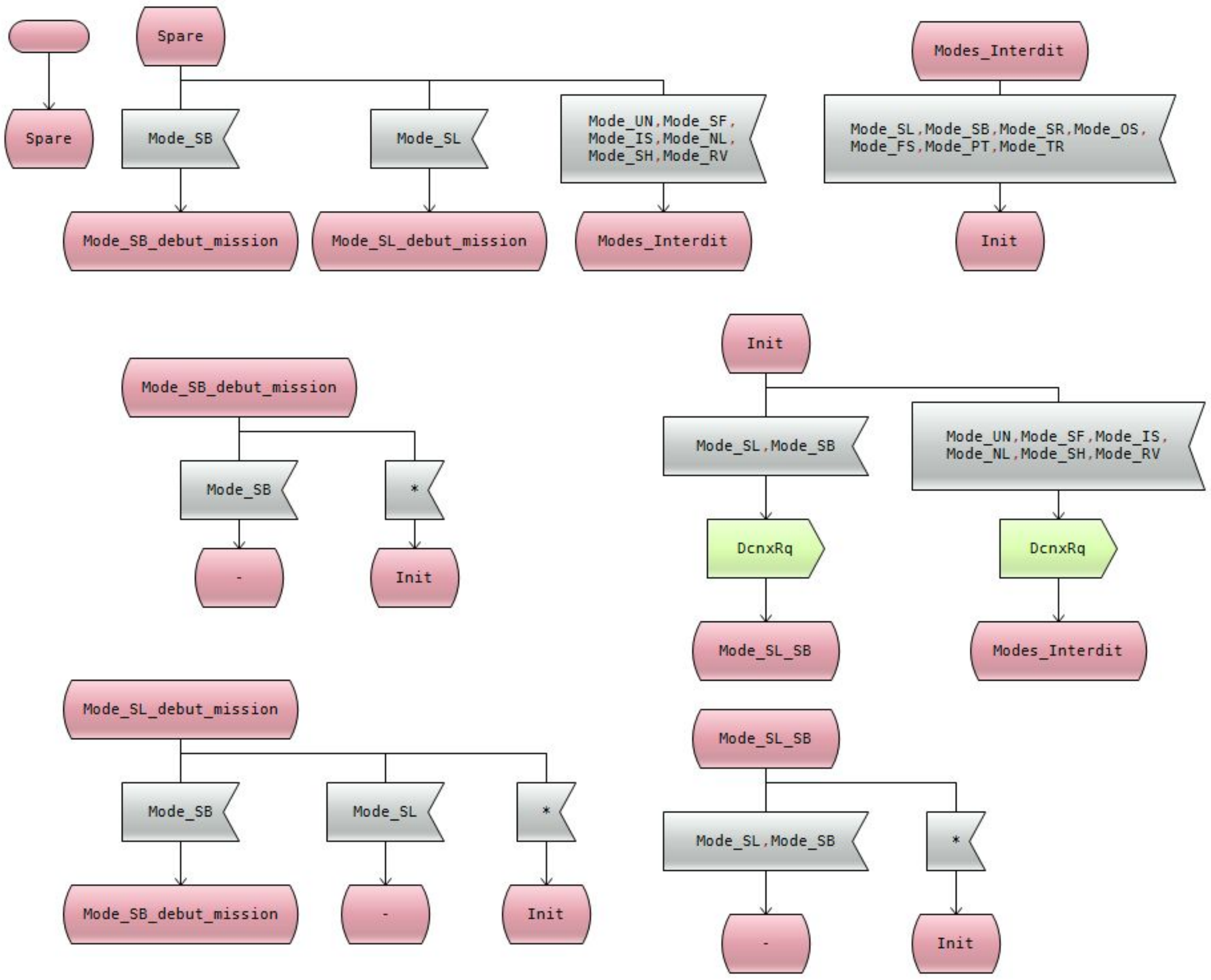


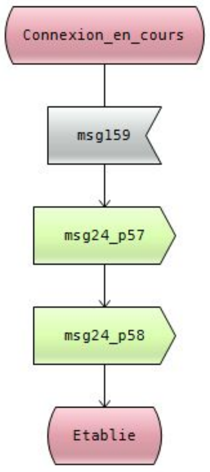
Figure 9 - “Disconnect depending on the mode” SDL state machine

In that example the main difference is on the inputs of the state machine. The Matlab model uses logical operators AND and OR to identify which input was received; the SDL model is just a list of inputs, and the star means any other input.

In both examples the model is equivalent from a functional point of view, depending on the reader’s technical background one or the other might be easier to read and understand.

Model verification

The experiment included some simulation of the SDL model with small prototyping graphical user interface in order to verify the behavior was correct. Once the model was considered correct, PragmaDev symbolic resolution tool, result of PragmaList [8][9] common lab, has been experimented on the model. This technology combines the transitions from a symbolic point of view, and tries to solve each possible combination like it would do with an equation. If there is a solution to the equation the path is possible. The first objective with that technology was to automatically generate the minimum number of test cases with a maximum coverage. After a few trials the tool could not reach two transitions in the model. A manual analysis rapidly concluded this use case did not allow one of the generic functions in the model to return the values required to reach these two transitions. Once this was settled test generation out of the model was



successfully experimented and 17 test cases covering all transitions were automatically generated.
Five properties have been written to be verified on the model. As for the experiment, the properties were actual pieces of the state machine written with another language. For example the first property verifies that when in state Connexion_en_cours, when receiving msg159 the state machine goes to state Etablie and not any other.
The symbolic resolution tool has been ran on the model with its properties for a few hours reaching a substantial depth of search, meaning a substantial number of transition combination. As a result, within this exploration perimeter the properties were satisfied.

How to link asynchronous models to synchronous models

During the experiment it has been established the SDL model was further away from the implementation than the Matlab one. Because of its asynchronous principles it was more of a functional view of the behavior and therefore closer to the requirements. This clearly validated the idea of having a high level asynchronous executable functional model to make sure the requirements are properly understood. The question was how to link this asynchronous approach to a synchronous one. It turned out an asynchronous model, including a test case, can easily be connected to a synchronous one. For example a synchronous input can be evaluated at each tick and when the value of the input changes it generates an asynchronous message (Figure 10). On the other way around an asynchronous message output can be converted to a clock based value.

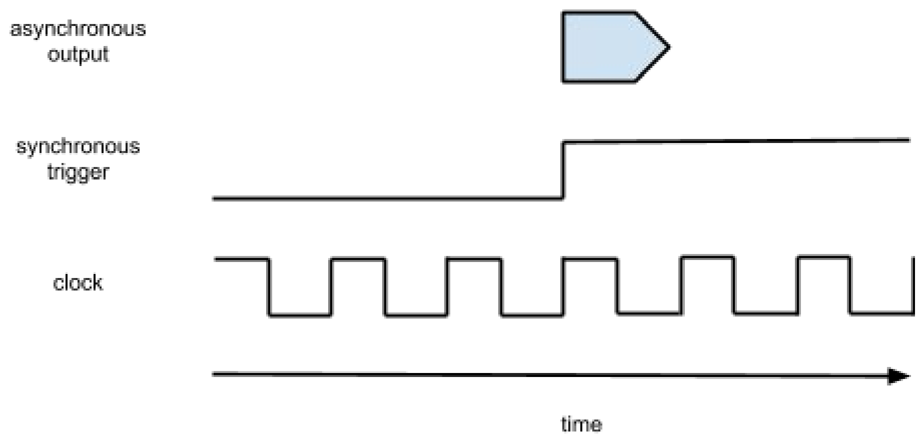


Figure 10: An synchronous change of value can be transformed to an asynchronous send

This shows that it would be possible to generate code out of the asynchronous model and connect to a synchronous target, or to generate test cases out of the asynchronous model and run them against a synchronous implementation in order to check it is conform to the model.

Conclusion & Future work

The experiment on this real use case in the railway domain has demonstrated that an SDL executable asynchronous model could be functionally equivalent to a Matlab synchronous model. Because of its asynchronous nature the SDL model is closer to the requirements, where a Matlab model is closer to the implementation. An SDL model could therefore be used by stakeholders early in the development process to formalize requirements and to verify them from a functional point of view. A Matlab model would still be used later on for the implementation. And the SDL model would be the reference to verify functional properties, or to generate test cases to verify the final implementation is functionally conform to the initial requirements.

Bibliography

[1] Flammini, F., “Railway Safety, Reliability, and Security: Technologies and Systems Engineering”, IGI Global, May 31, 2012 - Technology & Engineering - 487 pages.
[2] Liu, Jiang (et al.), “A Calculus for Hybrid CSP”, Programming Languages and Systems 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010 Proceedings, Springer LNCS 6461.
[3] Cimatti, Alessandro (et al.), “Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System”, Computer Aided Verification, 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings, Springer LNCS 7358.
[4] Callet, S., el Fassi, S., Fedeler, H., Ledoux, D. and Navarro, T. (2014) The Use of a “Model-Based Design” Approach on an ERTMS Level 2 Ground System, in Formal Methods Applied to Industrial Complex Systems (ed J.-L. Boulanger), John Wiley & Sons, Inc., Hoboken, NJ, USA.
[5] Le Sergent T., “SCADE A comprehensive framework for critical system and software engineering”, SDL Forum 2011, Springer LNCS 7083.
[6] International Telecommunication Union: Recommendation Z.100 (12/11) Specification and Description Language (SDL). http://www.itu.int/rec/T-REC-Z.100
[7] European Rail Traffic Management System ERTMS, http://www.era.europa.eu/Core-Activities/ERTMS/Pages/home.aspx
[8] Gaudin E., Deltour J., Faivre A., Lapitre A., “Model-Based Testing: An Approach with SDL/RTDS and DIVERSITY”. System Analysis and Modeling: Models and Reusability. 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings. Editors: Amyot, Daniel, Fonseca i Casas, Pau, Mussbacher, Gunter (Eds.). Springer LNCS 8769.
[9] www.pragmalist.org
[10] “Globally asynchronous locally synchronous”,Wikipedia, http://en.wikipedia.org/wiki/Globally_asynchronous_locally_synchronous
[11] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J. C. M. Baeten and S. Mauw, editors, Proceedings of CONCUR'99, volume 1664 of LNCS, pages 162-177. Springer, 1999.
[12] M. Mousavi , P. L. Guernic , J.-P. Talpin , S. Shukla and T. Basten "Modeling and validating globally asynchronous design in synchronous frameworks", Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, pp.384 -389.

TASTE IN ACTION

Maxime Perrotin[1], Konrad Grochowski[2], Marcel Verhoef[1], Damien Galano[1],
Michał Mosdorf[2], Michał Kurowski[2], François Denis[3], Estelle Graas[3]
[1]European Space Agency/ESTEC, Keplerlaan 1, 2201AZ Noordwijk, The Netherlands
[2] N7 Mobile Sp. z o.o., Łowicka 19/10, 02-574 Warsaw, Poland
[1]first.last@esa.int
[2]first.last@n7mobile.com
[3]first.last@ulg.ac.be

Abstract

In this paper, we present the results of the past years of active development and use of a set of tools named TASTE[1], which is developed under the supervision of the European Space Agency (ESA) since 2008. TASTE was created to facilitate the development of software using a combination of modern development techniques including formal methods, domain-specific languages and graphical editors to offer a nice user experience. Being free and favouring accessible APIs, TASTE is open and can be used in many areas, covering educational purposes as well as application on operational projects. We show how some of the TASTE core technologies, based on well-established languages, are actually used for the development of satellite instrument on-board software and documentation. As a technology demonstrator and lab experimentation platform, TASTE enables the investigation and exploration of techniques that are not commonly used in industry yet, such as functional programming, data and behavioural modelling, property checking, multi-core systems, etc.

The objective is to make better software using a solid engineering approach that does not stay stuck to the traditional “Text-based requirements translated to C source code” which is still mostly applied when developing embedded systems. But more than presenting a collection of technologies, this paper also explains our strategy and why a technical organisation such as ESA is probably the right one to lead the development of such tools and ensure their long-term support.

Keywords: SDL, ASN.1, AADL, MSC, VDM, Spark/Ada

Introduction

When creating software in the context of critical embedded systems, one of the first tasks is to set up a development plan with a clear strategy focusing on the priorities set by the customer and/or the end user of the system.

Priorities have to take a number of elements into account, for example requirements related to:

- Development schedule
- System safety
- System dependability
- Quality assurance and documentation
- Quality of the software code for performance, optimization
- Quality of the software code for readability, maintainability
- Reuse and automation
- etc.

In practice it is well known that all these requirements can not be covered at the same level of quality. Thus imposing a tight development schedule or low costs typically implies sacrificing some other aspects such as system analysis, robustness, testing or documentation. The question is then to know if it is possible to overcome this sacrifice and at what price. Is there any technology, tool, science that can support a software development process in such a way that perhaps by putting more effort in some places, it could improve significantly the overall software lifecycle as a side effect?

TASTE is a platform that was created with this goal in mind, and is used to explore and disseminate advanced software technologies in order to be ready for future space missions.

What are the systems that we target?

Today's processes for the development of classical Earth-orbiting spacecrafts are well mastered by our few European large system integrators: development is vastly done manually, but based on years of experience and reuse. The feedback we get is that each new satellite inherits from a lot of existing material and most of the time there is no need to change the approach while yet meeting most of the requirements seen above. It is therefore not these kind of well-established processes that are under focus: European-made classical satellites have simple, qualified software which need no major upgrade according to industry.

On the other hand, new upcoming space missions such as formation-flying systems, deep-space probes, robotic systems and next generation launchers usually are much more challenging. For example, do we know how to deal with distributed systems in space? Can we specify with lists of textual requirements all the complex scenarios to handle fault management? The companies that are involved in these new missions, now that the European Space Agency is open to many new countries, are more diverse today than in the past, and some have little background in the space domain and little software items to reuse. There are many areas where projects could benefit from dedicated tools and languages to approach new problems areas. For example, automatic code generation, which seemed underperforming and cumbersome a few years ago, is now mature and efficient enough to deal with software production. We need to characterize properly our expectations and find places where we could save time by automating repetitive tasks in order to put effort on more creative work.

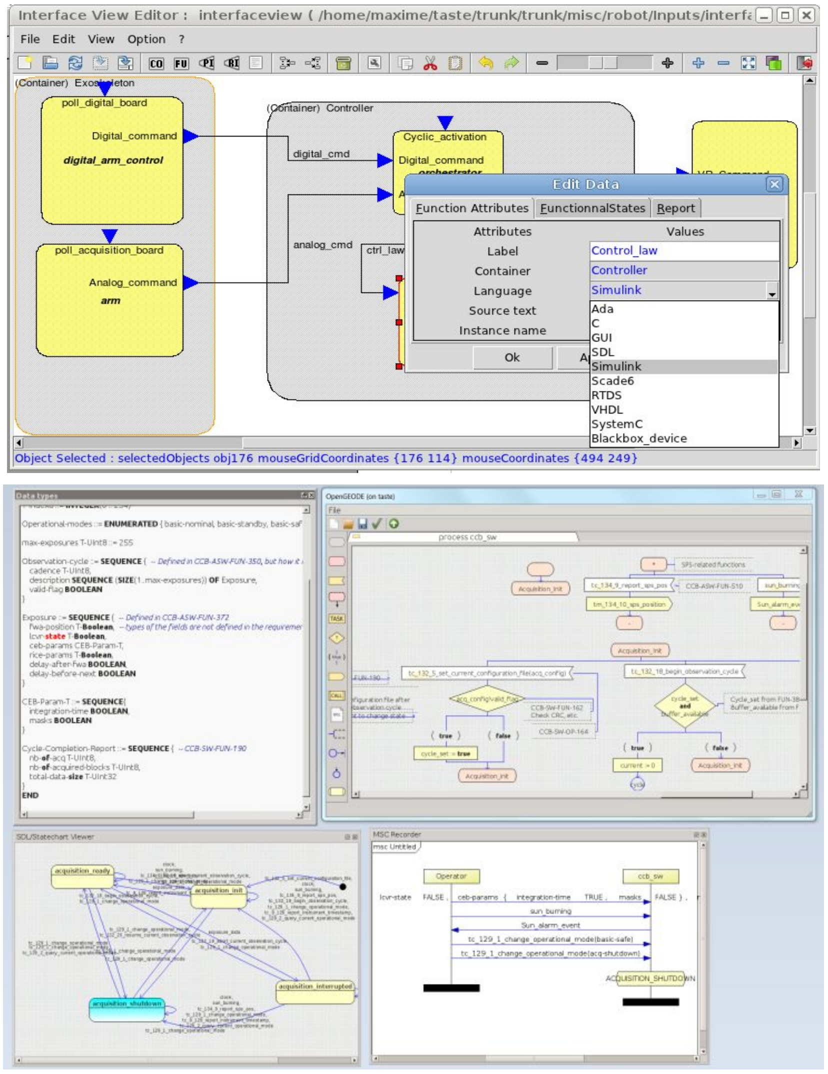
TASTE provides means to achieve this goal. By looking at the needs first, the idea is to find and easily adopt a good combination of technologies to make concrete and quantifiable progress in a software development lifecycle. By relying on solid, well matured technologies such as ASN.1, SDL and AADL, and by keeping the eyes open on various programming paradigms, with a long-term and open-source support in mind, TASTE is making the synthesis of the past 20 years of work in the field of model-based development, together with the recent trends in software programming using safer languages (VDM, Ada2012, Rust, F# and others). Of course, the results are not limited to space applications. Indeed, a safe and modern software development approach should in principle benefit all areas where the cost of a system malfunction is high and cannot be compensated by quick fixes after the deployment phase.

What have we done so far?

The development of TASTE started in 2008, and has been co-funded by ESA, industry, and universities. Many partners have participated and brought a lot of knowledge and material in the toolset. The overall strategy, vision and work plan is established by ESA.

In that context, a lot of work has already been achieved, and as a technology demonstrator, TASTE is already a full-featured set of tools, that goes very far in several areas of software development and deployment. The tool contains:

- a user-friendly entry point, containing a graphical editor for capturing a system architecture using the AADL language, and a state machine editor implementing partly the SDL language [3]



Attributes	Values
Label	Control_law
Container	Controller
Language	Simulink
Source text	
Instance name	

Object Selected : selectedObjects obj176 mouseGridCoordinates {176 114} mouseCoordinates {494 249}

```
Operational-modes ::= ENUMERATED { basic-nominal, basic-standby, basic-saf

max-exposures T-UInt8 ::= 255

Observation-cycle ::= SEQUENCE {  -- Defined in CCB-ASW-FUN-350, but how it
   cadence T-UInt8,
   description SEQUENCE (SIZE(1..max-exposures)) OF Exposure,
   valid-flag BOOLEAN
}

Exposure ::= SEQUENCE {  -- Defined in CCB-ASW-FUN-372
   fwa-position T-Boolean,  -- types of the fields are not defined in the requiremer
   lcvr-state T-Boolean,
   ceb-params CEB-Param-T,
   rice-params T-Boolean,
   delay-after-fwa BOOLEAN,
   delay-before-next BOOLEAN
}

CEB-Param-T ::= SEQUENCE{
   integration-time BOOLEAN,
   masks BOOLEAN
}

Cycle-Completion-Report ::= SEQUENCE {  -- CCB-SW-FUN-190
   nb-of-acq T-UInt8,
   nb-of-acquired-blocks T-UInt8,
   total-data-size T-UInt32
}
END
```

Fig. 1 - TASTE user-friendly GUIs

- a full, commercial-quality ASN.1 compiler dedicated to safety-critical systems, generating optimized Ada and C code, but also customizable documentation using template files
- technology to glue heterogeneous languages: user code can be written in C, Ada, Simulink, SDL, VHDL
- A custom integration of the Cheddar and Marzhin tools providing scheduling analysis of real-time systems
- a lot of other tools that ensure a correct-by-construction approach, allows to run simulations, perform regression testing and monitoring, create links to system databases, and much more.

Technical insight

Any complex system development requires the combination of many different disciplines to cover wide ranges of requirements. In terms of software, it is common to make use of a large number of technologies to address various aspects of the same problem:

- When coding manually, C and Ada are used to cover on-board software functionalities
- Assembly language is sometimes used for low-level code and boot software

- Script languages (Python, Tcl) are widely used to automate the testing of software
- Matlab-Simulink is frequently used by domain experts for the development and validation of control laws (to manage the position and trajectory of spacecrafts and rockets)
- VHDL or SystemC are used when hardware accelerators are needed (e.g. for image compression or in general on-board processing of science data)
- UML is sometimes used to draw informal sketches for documentation purposes
- SDL to model state machines with precise semantics and syntax
- etc.

The TASTE philosophy is to let application domain specialists spend most of their effort on their area of expertise and let tools automate the tedious parts which are of little interest to the system engineers. This means taking the wide variety of technology needed in practice, and putting everything together using tools, in order to ensure a system development that is correct by construction. It is not yet possible everywhere of course, but this is the goal we try to achieve. Correctness by construction covers different facets including this one: the provision of languages that are domain-specific allows to have one and only one way of addressing a problem - and rely on the research done in the area to prevent at the source the risk of a bad software implementation. To illustrate this facet, consider the following examples:

- Embedded systems are fundamentally based on state machines. Systems react on their environment and behave depending on their current mode of operation. Using a language such as SDL [4] in TASTE gives the right syntax and the corresponding checkers to ensure by construction that the application focuses on the state machine and nothing else.
- Data types: rather than reasoning in terms of physical encoding (size of integers, endianness), think about the range of values that are needed by the applications.

Even if it will need to be quantified, we expect that the generalized use of dedicated languages combined with tools automating software production, on the long term, will save a lot of time that is today spent on developing manually recurrent software functionalities.

User feedback

TASTE as a whole is only used by few companies. As shown in the case study presented in the next chapter, some of the TASTE components have already reached a high level of quality and usability and have proven excellent applicability to real-life projects ; but in this chapter we will discuss the braking factors and way forward to make sure that in the future, TASTE will be better known and used by many.

Taking the decision to adopt “unusual” technologies for a new project such as those present in TASTE (modelling tools, code generators, etc.) is always raising a number of questions:

- did anyone try them before in a similar context?
- is the technology really mature?
- what are the risks versus the benefits?
- is there a community of users?
- is there a commercial support available?
- is the underlying technology going to be supported on the long run?
- will someone be able to maintain the generated code in a satellite in 15 years?
- and by the way do we really want to automate the repetitive tasks that we have been doing manually for 30 years?

Even if some languages used in TASTE seem widely known in other domains of applications (e.g. ASN.1 or SDL are standard languages for telecommunication systems), they still represent small communities of users, and experience in computer science shows that technology is fragile and so pillars can collapse quickly. For example, UML that was called by many a “de facto standard” for modern software development is today declining fast and is abandoned by several tool vendors and users. From the point of view of non-technical decision makers, it can be difficult to see the difference between fashionable technologies which are designed for marketing purposes and technologies with solid grounds that can really help if used properly.

When the decision is taken, there is of course a price to pay. One part of this price is related to the effort needed to make an efficient use of the selected technology. Assimilation of knowledge requires effort and time. The techniques we are considering (state machines, abstract data types, model checking, etc.) are scientifically sound and mature, but at the same time they are often not widely known or understood by industrial developers. As a consequence there are limited useful resources and user feedback available so it is often perceived that the investment is risky due to little immediately visible benefits.

It is relevant to note that in the past few years, a significant interest has raised in software communities towards the functional programming style (declarative style, immutability, recursion, monads, etc.) These techniques are quite ancient but were never considered seriously for inclusion in mainstream, imperative programming languages. And yet they got a sudden and apparently unexpected second life, and many of these techniques are now included as first-class citizens in many languages (e.g. lambda functions). This simple example shows how software is a living and surprising area that is capable of evolving when technology proves an added value on real use cases.

One of the reasons ESA is investing a lot in software development is because exploration is part of its core mission. Exploration of space requires as a first step, as well as a side effect, exploration of new technology that is also useful on Earth. As a non-profit technical organization, ESA is not tight to any single tool, language or technology. ESA therefore has the freedom and the possibility to progress with a consistent and long term vision, taking the time to assess tools, make the right choices, and deliver results that do not depend on an immediate hit in market places.

ESA is looking at many possible ways of improving software development and in addition to TASTE has created projects named:

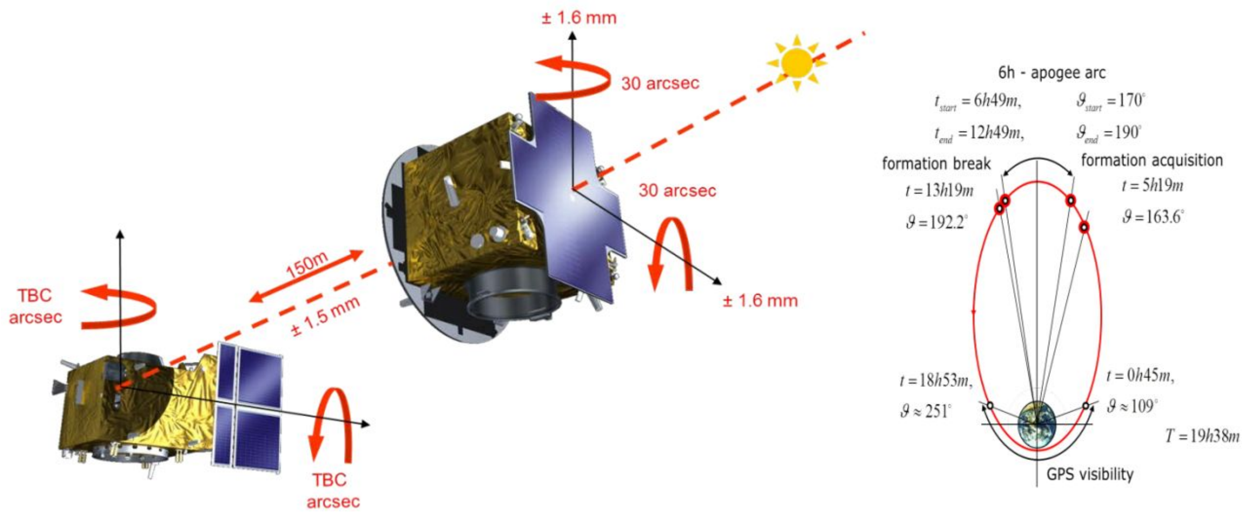
- SAVOIR and OSRA (trying to establish “reference architectures” of systems)
- COMPASS (exploring the fault detection mechanisms)
- EDS (modelling device driver interfaces)
- and a few others

In order to help these technologies get a chance to be used in operational projects and fly on future missions, ESA is working together with the European space industry to harmonize research and development initiatives and make communities talk together. Many technologies are driven by experts who get stuck in their area due to difficulty to have a global picture of the end user needs, to the lack of networks, connections with other research areas, or simply resources. For example, scientists who create formal methods usually have little time or interest in implementing end user interfaces that yet would be a key to make their results visible and usable by non-experts. ESA is capable of creating these missing links, animate working groups, experiment and disseminate tools, and that is the chosen direction, now that the level of trust in what tools like TASTE can do allows to go a step further.

Case Study - PROBA-3 Coronagraph Instrument Software

PROBA-3 is a mission devoted to the in-orbit demonstration of precise formation flying techniques and technologies for future ESA missions. It will fly an instrument named ASPIICS (Association of Spacecraft for Polarimetric and Imaging Investigation of the Corona of the Sun) as primary payload, making use of the formation flying technique to form a giant coronagraph capable of producing a nearly perfect eclipse, and allowing to observe the sun corona closer to the rim than ever before. The coronagraph system is distributed over two satellites flying in formation (approx. 150 meters apart). The so called Coronagraph

Spacecraft(CSC) carries the camera and the so called Occulter Spacecraft(OSC) carries the sun occulter disc.



± 1.6 mm
30 arcsec
30 arcsec
± 1.6 mm
150m
± 1.5 mm
TBC arcsec
TBC arcsec
6h - apogee arc
$t_{start} = 6h49m$, $\vartheta_{start} = 170^\circ$
$t_{end} = 12h49m$, $\vartheta_{end} = 190^\circ$
formation break
formation acquisition
$t = 13h19m$
$\vartheta = 192.2^\circ$
$t = 5h19m$
$\vartheta = 163.6^\circ$
$t = 18h53m$,
$\vartheta \approx 251^\circ$
$t = 0h45m$,
$\vartheta \approx 109^\circ$
$T = 19h38m$
GPS visibility

Fig.4 - PROBA-3 formation flying overview and orbit.

The proposed PROBA-3 Coronagraph System (ASPIICS) will be the first space coronagraph to cover the range of radial distances between 1.08 and 3 solar radii where the magnetic field plays a crucial role in the coronal dynamics, thus providing continuous observational conditions very close to those during a total solar eclipse, but without the effects of the Earth's atmosphere. ASPIICS will combine observations of the corona in white light and polarization brightness with images of prominences in the He I 5876 Å line.

ASPIICS will provide novel solar observations to achieve the two major solar physics science objectives: to understand physical processes that govern the quiescent solar corona, and to understand physical processes that lead to coronal mass ejections and determine space weather ([6]).

The PROBA-3 coronagraph optical design follows the general principles of a classical externally occulted Lyot coronagraph. The external occulter, hosted by the Occulter Spacecraft, blocks the light from the solar disc while the coronal light passes through the circular entrance aperture of the Coronagraph Optical Box (COB), accommodated on the Coronagraph Spacecraft (CSC).

COB, its control devices, and some other mechanisms form the Coronagraph Instrument. Following case study describes usage of some TASTE tools in development of Coronagraph Instrument Software (CISW).

CISW main responsibility is to integrate and control other instrument's subassemblies in order to achieve some level of autonomy from spacecraft, including automatic execution of all observation related operations. It is common that one of the most important part of software design is its interface, but in this situation (combining multiple subsystems into one single interface visible from main platform system) it is even more crucial to ensure proper integration of CISW with satellite, by enforcing consistency of data definitions between requirements, documentation and final implementation.

TASTE addresses those issues using language for modelling of data called ASN.1. This language was designed to ensure end-to-end data consistency: from a single description of data structures, tools can guarantee by constructions that semantically equivalent representations exist at any point in time in:

- software code,
- software documentation,
- system databases,
- test scripts.

It is also a well known standard, used for years in telecommunications, with various other tools available.

Requirement ID	Requirement Text	Verification
REQ-ID-0101	CISW shall implement the PUS service 6,2 “Load Memory using Absolute Address” to update data in RAM and EEPROM.	Test
REQ-ID-0102	CISW shall implement the PUS service 6,5 “Dump Memory using Absolute Address” to dump data from RAM, EEPROM and PROM.	Test
REQ-ID-0103	CISW shall implement the PUS service 6,9 “Check Memory using Absolute Address” to check data in RAM, EEPROM and PROM.	Test
REQ-ID-0104	CISW shall implement the PUS service 8,1 “Perform Function”.	Test
REQ-ID-0105	CISW shall implement the PUS service 17,1 “Perform a Link connection test”.	Test

Fig. 5 - Example of requirements for CISW TC.

Using this language it was possible to easily translate communication data related CISW requirements, like ones visible on Fig. 5 into ASN.1 model (Fig. 6). Such model is readable even by non-programmers and provides short step between textual requirements and verifiable model.

```
-- List representing all possible data structures contained by TC.
-- Only single item from this list can be present in TC at once.
-- Present item is determined by requested service type and sub-type.
CISW-TC-ApplicationData ::= CHOICE
{
  -- Data for PUS(6,2) request - Load Memory using Absolute Address.
  loadMemoryAbsolute         TC-PUS-6-2-LoadMemoryAbsolute,
  -- Data for PUS(6,5) request - Dump Memory using Absolute Address.
  dumpMemoryAbsolute         TC-PUS-6-5-DumpMemoryAbsolute,
  -- Data for PUS(6,9) request - Check Memory using Absolute Address.
  checkMemoryAbsolute        TC-PUS-6-9-CheckMemoryAbsolute,
  -- Data for PUS(8,1) request - Perform Function.
  performFunction            TC-PUS-8-1-PerformFunction,
  -- Empty placeholder for PUS(17,1) request - Perform Connection Test.
  connectionTest             Null
}
END
```

Fig.6 - ASN.1 description of some of the TC sent to CISW by the satellite platform

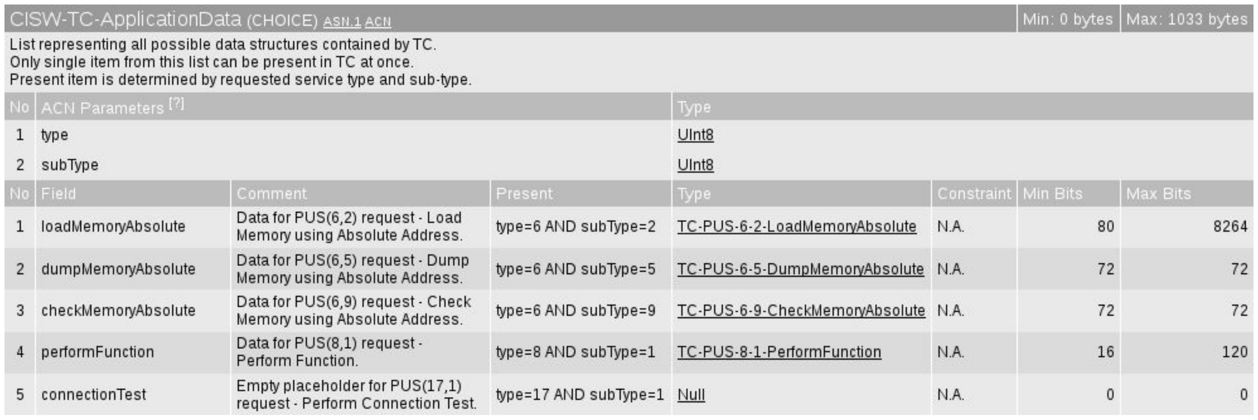
Although this model provides information only on logical components of data, it can easily be extended to describe physical representation. ASN.1 itself contains some standard encoding rules, like GSER, PER etc. It makes creation of data representation model very easy - just information about used encoding standards for already provided ASN.1 data models is needed. Yet in some situations, like with CISW, when some enforced encoding rules are outside of ASN.1 scope, standard solutions are not enough. Fortunately TASTE provides another language, ACN, which was designed for ESA with purpose of describing encoding rules for ASN.1 defined structures [5].

```
CISW-TC-ApplicationData<Base.UInt8:type, Base.UInt8:subType> []
{
  loadMemoryAbsolute           [present-when   type == 6    subType == 2],
  dumpMemoryAbsolute           [present-when   type == 6    subType == 5],
  checkMemoryAbsolute          [present-when   type == 6    subType == 9],
  performFunction              [present-when   type == 8    subType == 1],
  connectionTest               [present-when   type == 17   subType == 1]
}
END
```

Fig. 7 - ACN encoding rules for TC described in ASN.1.

Figure 7 shows, how previous ASN.1 model can be enhanced into complete data representation model by including of ACN encoding rules. With that addition, model contains enough information to be a source for generated detailed documentation, or encoding and decoding procedures for production code.

In CISW development the ASN.1 compiler named asn1scc [5] was used, which started as part of TASTE (and it is still included in it) but now it is a standalone tool. It is being used to ensure consistency between ASN.1 model (based on requirements), flight and testing code (some C functions generated from model), and required software interface documentation (partially generated from model). Consistency is achieved by treating ASN.1 and ACN model files as source code with asn1scc calls being included in build process. Same files are also used to generate documentation (Fig. 8), which will become part of Interface Control Document (ICD) - required document, which shall allow other components' providers to interface with CISW. ASN.1 and ACN files will be included in that document as attachments (together with generated C header files), allowing their reuse by document receivers.



CISW-TC-ApplicationData (CHOICE) ASN.1 ACN	Min: 0 bytes	Max: 1033 bytes

List representing all possible data structures contained by TC.
Only single item from this list can be present in TC at once.
Present item is determined by requested service type and sub-type.

No	ACN Parameters [?]	Type
1	type	UInt8
2	subType	UInt8

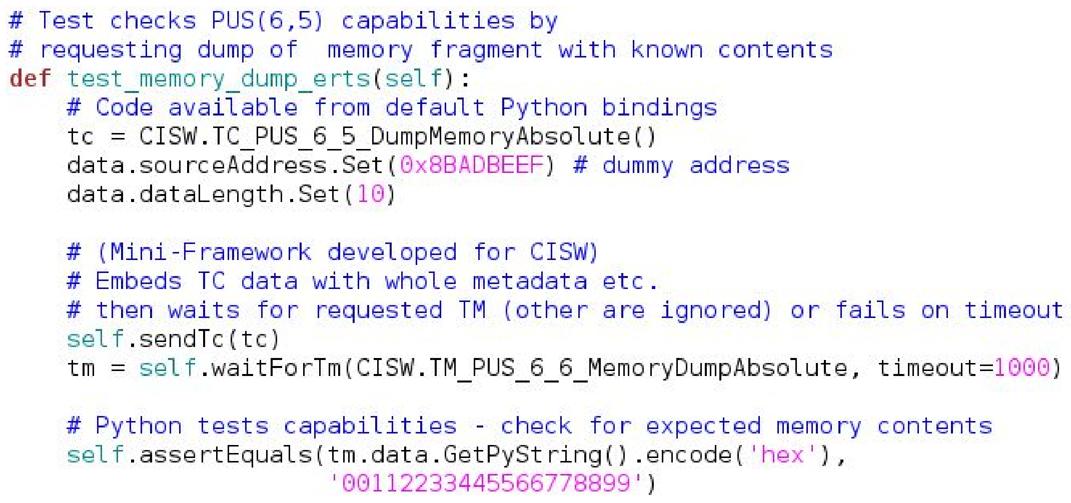
No	Field	Comment	Present	Type	Constraint	Min Bits	Max Bits
1	loadMemoryAbsolute	Data for PUS(6,2) request - Load Memory using Absolute Address.	type=6 AND subType=2	TC-PUS-6-2-LoadMemoryAbsolute	N.A.	80	8264
2	dumpMemoryAbsolute	Data for PUS(6,5) request - Dump Memory using Absolute Address.	type=6 AND subType=5	TC-PUS-6-5-DumpMemoryAbsolute	N.A.	72	72
3	checkMemoryAbsolute	Data for PUS(6,9) request - Check Memory using Absolute Address.	type=6 AND subType=9	TC-PUS-6-9-CheckMemoryAbsolute	N.A.	72	72
4	performFunction	Data for PUS(8,1) request - Perform Function.	type=8 AND subType=1	TC-PUS-8-1-PerformFunction	N.A.	16	120
5	connectionTest	Empty placeholder for PUS(17,1) request - Perform Connection Test.	type=17 AND subType=1	Null	N.A.	0	0

Fig. 8 - Documentation (in HTML format) generated from ASN.1.

This functionality alone proves usefulness of ASN.1 modelling with TASTE tools, but, as usual, it turns out that once model is prepared a lot of new possibilities of its usage appear. CISW developers decided to use asn1scc and other TASTE Data Modelling Tools (DMT) in their internal software verification process, to ensure higher code quality. At first asn1scc was used to generate encoder and decoder unit tests based on data model. Those tests help achieve high code and functionality coverage with close to zero additional development costs. Still, highly regarding code (product) quality, development team decided to invest some time to create own test framework based on possibilities provided by TASTE.

DMT allows to create Python and SQL (among many others) bindings for structures described in ASN.1. In CISW testing SQL bindings are mostly used to log and later analyze telemetry reported by working software. Each received message can be stored in database, in decoded form, allowing execution of complex queries - for example to calculate statistic of reported housekeeping parameters. Work needed to enable SQL bindings was minimal, assuming Python bindings already existed. With them many additional tools (including analytical) can be used by developers. At this moment they're used for manual tests, but probably, together with code development, some automatic integration tests will be created and connected to internal verification test suite.

Python bindings turned out to be an essential help in development process. Basic bindings come at zero costs, but embedding them in build process, providing some utility functions and glue code for additional modules required some moderate amount of work, but at this moment it seems it was a justifiable invest.



```
# Test checks PUS(6,5) capabilities by
# requesting dump of  memory fragment with known contents
def test_memory_dump_erts(self):
    # Code available from default Python bindings
    tc = CISW.TC_PUS_6_5_DumpMemoryAbsolute()
    data.sourceAddress.Set(0x8BADBEEF) # dummy address
    data.dataLength.Set(10)

    # (Mini-Framework developed for CISW)
    # Embeds TC data with whole metadata etc.
    # then waits for requested TM (other are ignored) or fails on timeout
    self.sendTc(tc)
    tm = self.waitForTm(CISW.TM_PUS_6_6_MemoryDumpAbsolute, timeout=1000)

    # Python tests capabilities - check for expected memory contents
    self.assertEquals(tm.data.GetPyString().encode('hex'),
                      '00112233445566778899')
```

Fig. 9 - Example of testing capabilities available via Python bindings.

Figure 9 shows how simple testing complex capabilities of CISW becomes with tests written in Python. All encoding and decoding is done by bindings generated by TASTE. Only connection methods were implemented by hand. Additional gain is that this test can be used with many different real connections - starting from unit tests, where whole CISW is embedded inside tests executable (no real connection is used), through connecting to simulator and ending with real serial connection to development board with uploaded CISW. This way the same test can be used by developer on his/her machine and for integration tests with real hardware. Such capabilities greatly enhance both development speed and code quality, which is a rare combination, worth additional work.

Working with TASTE proved that one of its biggest advantages is that it is a completely open source project. For example, at some point it turned out, that C code generated by asn1scc produces some warnings in older versions of C compiler. Those warnings were superfluous and not present in newer versions of compiler, but old version was required by CISW target platform toolset. As CISW team aims at warning free code, this could become an issue, yet it was easily fixed by making a simple change in asn1scc (renaming some generated variables names). Not only it was possible ad hoc due to open source nature of TASTE, it was also easy to provide that change back to community and current version has this issue already fixed.

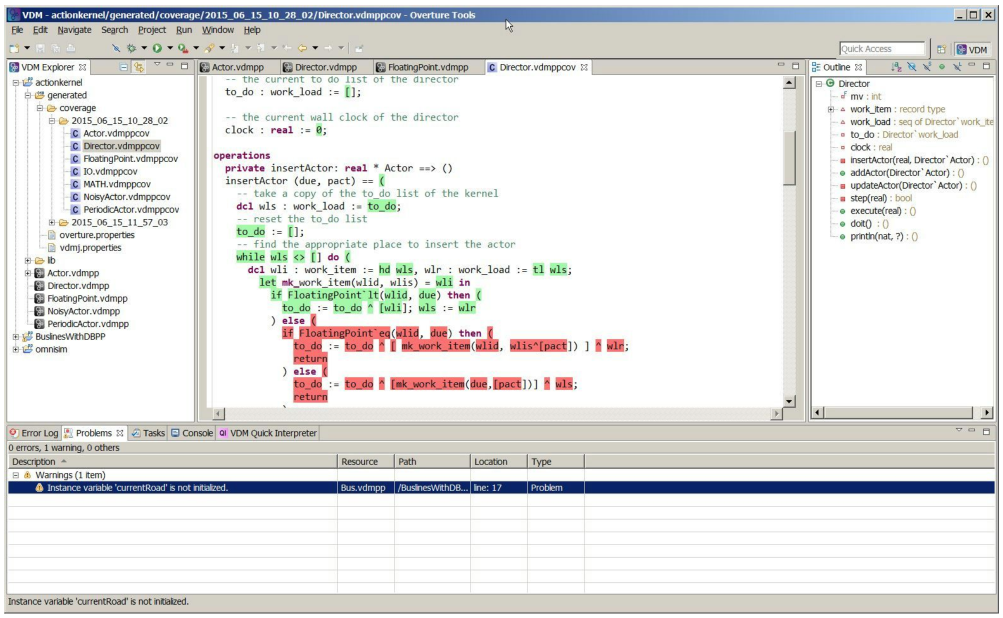
CISW is still being developed, so it is hard to predict how much impact TASTE will finally have on this project, but at this moment it proved to contain useful engineering tools, which can greatly help in assuring quality, safety and maintainability of code. Those tools are more than just technology demonstrator. Without them development process could not be at the same level of advancement at this point in time and they became its crucial parts.

Conclusion and future

TASTE is continuously being developed; currently several areas are being explored:

- Co-simulation at system level
- Hardware-software co-design
- Traceability and links to requirement engineering
- Supporting multi-core architectures
- Scheduling analysis, measurement of worse-case execution time
- Generation of Spark/Ada 2012
- Model checking and verification of properties
- Integration with formal methods such as VDM and the Overture toolset [2]
- Work on other domains of applications such as robotics systems (compatibility with environments such as ROS or ROCK)

There are other efforts in industry to cover some of these areas. Tools for system engineering, user requirement notation, use case maps etc., are all of interest for the future of the TASTE approach.


VDM - actionkernel/generated/coverage/2015_06_15_10_28_02/Director.vdmppcov - Overture Tools

```
actionkernel
  generated
    coverage
      2015_06_15_10_28_02
        Actor.vdmppcov
        Director.vdmppcov
        FloatingPoint.vdmppcov
        IO.vdmppcov
        MATH.vdmppcov
        NoisyActor.vdmppcov
        PeriodicActor.vdmppcov
      2015_06_15_11_57_03
      overture.properties
      vdmj.properties
  lib
  Actor.vdmpp
  Director.vdmpp
  FloatingPoint.vdmpp
  NoisyActor.vdmpp
  PeriodicActor.vdmpp
BusinesWithDBPP
omnisim
```

```
  -- the current to do list of the director
  to_do : work_load := [];

  -- the current wall clock of the director
  clock : real := 0;

operations
  private insertActor: real * Actor ==> ()
  insertActor (due, pact) == (
    -- take a copy of the to_do list of the kernel
    dcl wls : work_load := to_do;
    -- reset the to_do list
    to_do := [];
    -- find the appropriate place to insert the actor
    while wls <> [] do (
      dcl wli : work_item := hd wls, wlr : work_load := tl wls;
        let mk_work_item(wlid, wlis) = wli in
          if FloatingPoint`lt(wlid, due) then (
            to_do := to_do ^ [wli]; wls := wlr
          ) else (
            if FloatingPoint`eq(wlid, due) then (
              to_do := to_do ^ [ mk_work_item(wlid, wlis^[pact]) ] ^ wlr;
              return
            ) else (
              to_do := to_do ^ [mk_work_item(due,[pact])] ^ wls;
              return
```

```
Director
  mv : int
  work_item : record type
  work_load : seq of Director`work_ite
  to_do : Director`work_load
  clock : real
  insertActor(real, Director`Actor) : ()
  addActor(Director`Actor) : ()
  updateActor(Director`Actor) : ()
  step(real) : bool
  execute(real) : ()
  doit() : ()
  println(nat, ?) : ()
```

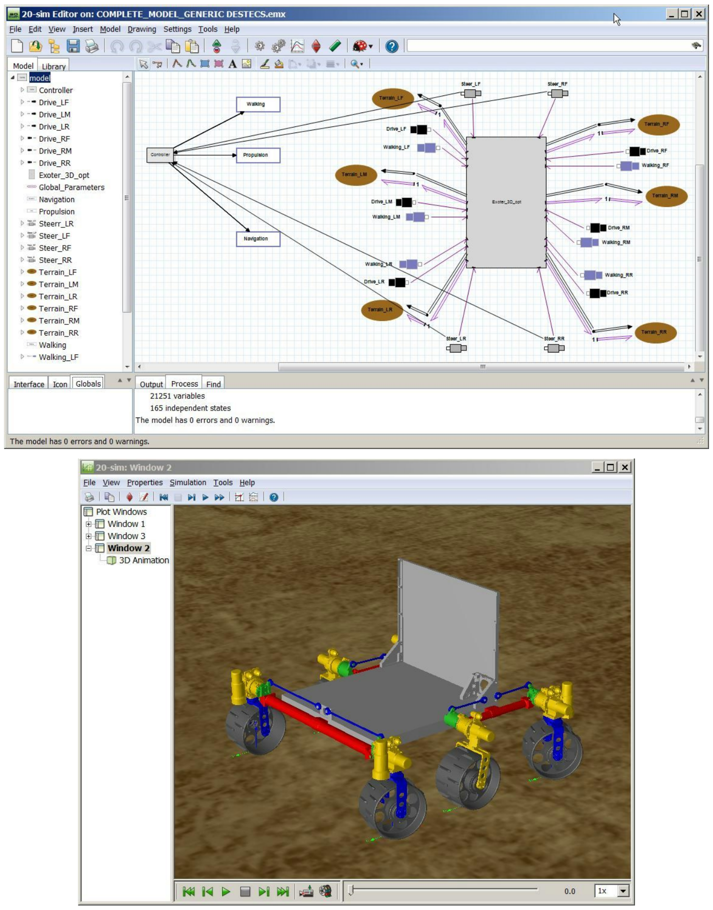
0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
Instance variable 'currentRoad' is not initialized.	Bus.vdmpp	/BuslinesWithDB...	line: 17	Problem

Instance variable 'currentRoad' is not initialized.

Fig. 10 - Overture showing coverage information after a validation experiment

TASTE was created with the objective to develop software with a more scientific mindset than the classical edit-compile-debug approach. To go further in that direction, some work is being done for example with formal methods, in particular the suite of VDM languages [2,7]. It offers a very powerful approach to specification and implementation of software; it is used in several industrial areas but is mostly unknown to the space community. In the spirit of functional languages, but not limited to that paradigm, it provides a rich and compact syntax to express complex requirements. It forms the basis for formal analysis as well as pragmatic support for testing and validation. On the other hand it offers interactive prototyping and testing, proof support and model checking, which is well suited to the verification of properties when using domain-specific languages. Combining all these technologies in a consistent way might help making a real step forward in the quality of software. This approach to model-based systems engineering tries to build upon the results from the domain of cyber-physical systems of systems, as promoted by the DESTECS INTO-CPS project [8,9]. In this project, we couple abstract continuous time models of the environment, specified in Bond graphs, with discrete event models of the controller (spacecraft) using VDM, which allows validation of system properties very early in the life-cycle, for example by means of 3-D visualisation, as is shown in Figure 11. Currently, an early prototype is available where SDL models in TASTE, specifying the reactive systems behavior in terms of state machines, are enriched with VDM models in Overture [2] to express the algorithmic parts of the system behavior, whereby ASN.1 is used as the vehicle to exchange data types between these formal techniques.



```
21251 variables
165 independent states
The model has 0 errors and 0 warnings.
```

```
The model has 0 errors and 0 warnings.
```

Fig. 11 - Multi-domain co-simulation of a Mars rover using Overture, developed in DESTECS [8]

References

[1] TASTE project : http://taste.tuxfamily.org/

[2] VDM and the Overture project: http://overturetool.org/

[3] OpenGEODE, http://www.opengeode.net

[4] SDL: www.sdl-forum.org

[5] ASN.1 and ACN: http://taste.tuxfamily.org/wiki/index.php?title=Technical_topic:_ASN1SCC_-_ESA%27s_ASN.1_Compiler_for_safety-critical_embedded_platforms

[6] “Design status of ASPIICS, an externally occulted coronagraph for PROBA-3”, *Proc. SPIE* 9604, Solar Physics and Space Weather Instrumentation VI, 96040A (September 28, 2015); doi:10.1117/12.2186962; http://dx.doi.org/10.1117/12.2186962

[7] John Fitzgerald, Peter Gorm Larsen, Marcel Verhoef, “Collaborative Design for Embedded Systems - Co-modelling and Co-simulation”, Springer, 2014, ISBN 978-3-642-54117-9

[8] The DESTECS project, http://www.destecs.org

[9] The INTO-CPS project, http://into-cps.au.dk

Designing, developing and verifying interactive components iteratively with djnn

Stéphane Chatty Mathieu Magnaudet Daniel Prun
Stéphane Conversy Stéphanie Rey Mathieu Poirier
Université de Toulouse - ENAC
7 av. Edouard Belin, 31055 Toulouse, France
{firstname.lastname}@enac.fr

ABSTRACT
Introducing iterative user interface design methods into the development processes of safety-critical software creates technical and methodological challenges. This article describes a new programming paradigm aimed at addressing some of these challenges: interaction-oriented programming. In this paradigm any piece of software consists of a hierarchical collection of components that can interact among themselves and with their environment, and its execution consists in propagating activation through interactions between components. We first describe the principles of interaction-oriented programming, and illustrate them by describing the basic components provided by the djnn programming framework to create interactive software. We then show how interactive programming provides a basis for formulating and checking properties that capture requirements on interactive components. The rest of the article is dedicated to example design and development scenarios that illustrate how development environments could leverage interactive programming in the future so as to jointly address the requirements of modern user interface design and safety-critical software development.

ACM Classification Keywords: H5.2 Information Interfaces and presentation: User Interfaces; D2.2 Software Engineering: Design Tools and Techniques; D3.3 Programming Languages: Language Constructs and Features.

Author keywords: interactive software, design process, development process, interactive component, software verification

INTRODUCTION

There are well established methods for developing safety critical software, such as those prescribed in the DO178 standards. In some domains, solutions have been developed to apply these methods to user interfaces as well. For instance, in aeronautics the ARINC 661 standard defines a collection of well known interactive components (or widgets), such as lists and menus. The protocol of these components is defined in such a way that they can be developed and certified individually, then reused at will. Industrial tools have even been introduced to assemble them graphically, then generate the corresponding code.

However, with ongoing plans to introduce more modern user interfaces in these safety critical settings, new questions appear. Not only are the envisaged interactive components often more complex and more difficult to specify than their predecessors, there is also a trend toward interface customization. Each aircraft or car manufacturer wants to have its own distinctive signature, in terms of interaction and not only graphical appearance. Consequently, they expect equipment providers to deliver products whose behavior and appearance can be modified. This means that one cannot rely solely on industry standards that define interactive components, and that methods must be proposed for designing and developing custom components, usually as a collaboration between an equipement provider and an integrator.

The user interface industry has developed and validated methods for designing custom interactive software, and recent R&T projects have shown that they can be applied successfully by the aeronautics industry. However these methods involve various actors, including human factors experts and designers, and are iterative by nature. Integrating them in industrial development processes brings new challenges, and appropriate tool chains are not yet available to support this. In addition, the state of the art in interactive software development has not yet reached the same level of maturity as other branches of computer science. Interactive software is still complex and costly to produce [18, 19], and model driven engineering methods are not yet widely used. This does not provide a favorable context for developing and validating custom components on a repeated basis. In particular, validating interactive software that contains code written in a traditional programming language is very costly.

In this article, we propose a theoretical and practical contribution towards the resolution of these two issues: the convergence of user interface design methods and critical software development processes, and the validation of custom interactive software. This contribution consists of a software execution model and a software architecture, whose combination provides the basis of what we call interaction-oriented programming. This allows to organize interactive software as a collection of components whose execution can be analyzed and whose definition can be incrementally refined. The proposed execution model and component architecture are implemented in a development framework named `djnn`, that has been used successfully in various projects.

We first analyze reasons why it is important to account for interactive software design methods. Doing so, we outline requirements for future design processes and tools that would combine the needs of interactive and safety critical systems. We then stress the key role of software architecture and execution models in fulfilling these requirements. After reviewing the state of the art on these topics, we introduce a theoretical framework for interactive components that defines a software architecture and an execution model. We then describe the djnn framework and use a simple aeronautical component (a primary flight display) and a series of idealized development scenarios to illustrate how new processes and tools for designing interactive software can be derived from this work.

ANALYSIS: WHY IS INTERACTIVE SOFTWARE SPECIAL?

Managing hidden requirements

One of the keys to successfully developing complex systems is the management of requirements. Gathering requirements, analyzing them, and tracing them in products represent an important part of the development effort. Various development methods and tools have been proposed for this purpose, and for supporting certification processes. Unfortunately, user interfaces suffer from a fatal flaw with this regard: requirements are impossible to gather in advance. There are hidden requirements that keep appearing during design and development processes, or even during the use of the final systems. As an example, consider an equipment in which two critical information fields are shown side by side. The two graphical representations may work perfectly when tested independently, and when using them together visual interferences or perceived inconsistencies between them can lead users to errors in reading them. It can also be discovered later that one of these representations does not work well when users are in particular mental conditions that had not been anticipated. Some of these emerging requirements, when not caught early enough, can be palliated by operational procedures or user training. Others can lead to safety flaws, or to outright rejection of the system by its intended users.

A day might come when cognitive sciences provide us with enough knowledge that requirements can actually be gathered in advance, but this is at best a long time goal. Until then, the best known solution for securing requirements is iterative design, a flavor of agile methods developed specifically for interactive software. Users are presented prototypes created by designers and usability experts, new requirements emerge from the confrontation, and new prototypes can be designed. During this process, two collections are incrementally created: a collection of design elements, and a collection of requirements. It is only after a number of iterations that the two collections can be considered sufficient.

This iterative process is not intrinsically incompatible with the processes used in safety-critical developments. Nevertheless, not only does it require dedicated tools for the initial phases of design, it also interferes with further development phases. In theory, user interface design could be done prior to actual software development, so that the two processes are independent. In practice however, design generally continues in parallel with development, if only because hidden requirements keep showing up. This means that solutions must be proposed to manage how the two processes interact with each other.

Architecture of interactive software

The above provides a first set of requirements for tools dedicated to user interface design in critical systems: an architecture that allows the incremental refinement of design elements and their execution, as well as the incremental definition of requirements and their checking. The desire to create customizable user interfaces highlights additional requirements: the ability to separately define the behaviors of given components, and check that the resulting user interfaces still meets the requirements.

Interoperability and interchangeability of software components is actually a universal concern, and it is the essence of software architecture. There are various situations in which programmers need to change how components are combined in their software. This includes early design phases, in which prototypes are developed. And this also includes various refactoring phases, whether during initial development or when the software is reused to create new products. Each programming paradigm provides support for the interoperability of a certain type of components, and therefore for software architecture. For instance, functional programming makes it easier to refactor software that performs computations. Similarly process algebra makes it easier to refactor software that reacts to input.

However, interactive software brings its own class of refactoring. There are many different ways to design a user interface for a given task. Take for instance the single pressing of a button to activate a function. The button can just change state visually, for instance by changing color. This atomic change can be expressed through an assignment, either of the color itself or of a symbolic state of the button. Alternatively, the button can change with an animation. This would require the state assignment to be replaced with a process that repeatedly modifies the visual state, and that can continues in parallel with further interactions. These two options are profoundly different in existing programming paradigms, and this makes refactoring costly. To compound matters further, user interfaces can make use of various interaction modalities such as graphics, sound, touch input, speech input, eye tracking, etc. Not only is each modality different from the others, they can also be combined in ways that require to manage interaction state, time intervals, and other complex execution patterns.

Various architecture patterns have been proposed to support the interoperability of components in interactive software. As exemplified above, some situations are easily expressed by simple constructs in existing programming languages, such as function calls, assignments or loops. Others require specific solutions such as events, data-flows, state machines. This diversity of control structures and component interconnection mechanisms, if not derived from a reduced set of primitives, hampers interoperability. It also limits the formulation of an execution semantics for programs, and therefore the ability to formulate and check properties on software components.

Another consequence is in the complexity for programmers. Each individual solution alleviates one source of complexity, for instance state machines make it easy to manage state dependencies. But as soon as several solutions are mixed, new sources of complexity appear. For instance, languages such are QML must be combined with traditional languages such as C++ to produce full applications, thus reintroducing in programs the traditional architecture patterns and control structures. Similarly, combining state machines with dataflows can rarely be performed without writing additional code in a traditional programming language. Programmers therefore end up manipulating independent concepts that each describe part of the program execution, and that do not have a clear common base.

Therefore, defining a minimal set of primitives from which the behavior of interactive software can be derived is required both for raising interactive software to the same level of manageability as other types of software, and for supporting development processes that rely on software refactoring.

STATE OF THE ART

Complexity

Various methods have been proposed over the last decades to make interactive software development less complex. The initial approach, mostly in the 1980s and 1990s, consisted in creating User Interface Management Systems and user interface toolkits on top of traditional programming languages. This approach evolved into two areas of research. On the one hand were collections of reusable interactive components such as dialogue boxes and buttons, that evolved into specialized languages and standards such as XAML, XUL, QML and ARINC 661.

On the other hand were software patterns aimed at managing the architecture requirements of interactive software, that differ significantly from those of computation-oriented software. Some of these patterns were aimed at separating the interactive code from the computation-oriented and data-oriented code. This includes the Seeheim and ARCH architecture, and the MVC, PAC and MVVM patterns. Other patterns were aimed at providing better support for the execution and control patterns encountered in interactive software. This starts with callback functions and the Inversion of Control pattern that account for the prevalence of external control in interactive software. But this also includes more complex patterns to support state management (state machines, hierarchical state machines, Statecharts) and dataflow (one-way constraints, functional reactive programming [11], dataflow bricks [6]). However, as mentioned previously, these solutions provide only local complexity relief: they make some parts of the behavior easier to formulate, but the overall complexity remains high because of how heterogeneous constructs are combined.

Development process

Historically, two approaches have been taken to develop graphical user interfaces. The first consists in giving a functional specification to programmers, and rely on their graphical skills. The second, used when visual quality is desirable, consists in asking graphical designers to produce visuals then asking programmers to reproduce them in their programs. Having programmers recode the graphics like this is both a waste of time and a cause of errors.

In contrast with this, traditional programmers can split their tasks and work in parallel, relying on well supported integration techniques such as separate compilation and linking. Interactive software developers could take advantage of similar solutions. First, they could to split their tasks and work in parallel with graphical designers. Then, they could also split tasks during iterative design phases with users and human factors specialists. Solutions have been proposed for this, leveraging on the similarities between the visual and logical structure of a user interface and the abstract syntax tree in traditional programming [7]. Considering graphics and interactive behaviors as nodes of a tree allows to produce them independently, then build the tree during a loading phase at compilation or execution time, and execute the resulting tree. This has been shown to yield significant improvements both in terms of effort and development time span[**?**]. However, this is only feasible for those parts of the software that can be consistently modeled as nodes in the execution tree. To be fully operational, this approach requires that 1OO% of programs can be modeled in the same framework.

Another approach consists in applying model-driven engineering methods to interactive software. In this approach, domain experts produce an abstract model of the task to be carried out by users, then all or part of the software is derived from this abstract model. However this only work for some classes of user activities, and the resulting user interfaces are stereotyped and of limited usability. More practical variants of this approach have also been tested, where the user interface is expressed using a theoretical model, but it is produced through a design process rather than generated from a task model. This allows to take advantage of the benefits of model-driven architectures without losing the interface quality brought by the design process. For instance in the PetShop system, applications are created by combining Object Petri Net models with Java code [20]. However, like above this approach will reach its full potential only when 100% of the user interface can be expressed in the model.

Code verification

During the last decades, various methods dedicated to the verification of interactive system properties have been proposed. The widely used approach relies on the test of the final system (or a prototype of it) where end-users accomplish selected tasks in a dedicated environment. Observations and measurements performed during the execution are used to assess whether the system fulfills the expected properties or not. The main drawback of this approach lies in its lack of exhaustivity because properties cannot be checked against all possible executions of the system. Moreover, this approach can only be used after the development (the system must be available) which constitutes another negative point: bugs are way more expensive to locate and correct at this development stage.

Model-based methods have been proposed to minimize these issues: at design time, a model of the future system is built and properties are verified by studies performed on the model rather than on the final system itself. The underlying logic behind this approach is that if a property holds on the model, and if the final system is built according to the model, then the property holds on the final system. Model checking aims at verifying properties on the state-transition structure built during the simulation of the model [20]. This is very similar to methods used for safety-critical software (eg. Esterel: [4], Scade: [12]). Alternatively, proof-based methods consist in mathematically proving the preservation of properties (invariants, pre-conditions and / or post-conditions) during successive refinements of the model (VDM: [10], Z: [16] or B: [2]).

Although they have been shown to be very effective, such model-checking approaches suffer from their impossibility to encompass systems with infinite number of states or transitions which is a major limitation in the context of interactive systems. Moreover, the assumption that "the final system is built according to the model" is hard to reach for reasons explained previously: no available model describes 100% of a user interface. To palliate this, some authors introduce a simplified model called the abstract user interface, in which the totality of a user interface can be described. Properties can be checked on this model. However, the process of converting this into a concrete user interface by adding code to the abstract user interface limits the benefits that can be obtained from the model based approach.

Abstract interpretation ([9]) is a verification method based on static analysis of the software code. An abstract semantic is extracted from the code and can be used to verify properties and perform optimisations [1]. This approach has historically been used for the verification of various properties of programs. For instance, the static analyzer Astrée is able to prove the absence of some types of run time errors on C programs ([5]), and has been used in safety-critical projects. For interactive properties, [17] first proposed this approach with the objective of providing a unification canvas for verification techniques. [14] described the verification of interactive Web pages through the static analysis of the user interface with ergonomic rules, encoded in UsiXML. [21] proposed to build and exploit a graph-oriented semantics of an interactive device to support the verification of properties. In this work, an existing device was analyzed and modeled with a graph whose arcs represent user actions and nodes observable states. It was then possible to compute some interactive properties on the graph that can be interpreted at the device level.

Abstract interpretation is an efficient application of the model checking approach: work is performed directly at the lowest level, that is the code of a concrete user interface, and therefore avoids the limitations of using an abstract model. Paradoxically, its main drawback for interactive systems lies in the impossibility to access to higher levels of description of the interface, such as those contained in the abstract user interface, because they usually have disappeared during implementation and compilation. It becomes impossible to check properties that would be expressed at these higher levels, such as "is this rectangle red when this button is pressed?". When the software is built with languages that do not capture the appropriate level of information (e.g. C ot Java), the missing information must be introduced by producing a model by hand.

[1]Concrete semantics are mathematically well defined objects that explicit the meaning and the possible behaviors of the program. Concrete semantics are generally not computable, which makes all non-trivial properties undecidable. To avoid this, abstract semantics are introduced as computable approximations of concrete semantics in which more properties are decidable.

Analysis and positioning

Most limitations of the state of the art derive from the same cause: the limited power of expression of the patterns, languages and models available. Each of the available solutions for expressing parts of interactive software have focused on a given requirement that was not fulfilled by traditional programming languages: supporting external control, or providing reusable dialogue boxes, or managing state, etc. Each alleviates one source of complexity in interactive software, but none provides a complete solution like modern programming languages do for computation-oriented software. This has consequences on software complexity, on development processes, and on the ability to verify code properties.

In this article, we describe an alternative solution that overcomes these limitations using an approach similar to Lustre or Esterel [15, 4]: adopting an execution model dedicated to the category of software that is being developed, and using it to develop the totality of the concerned software. Like Lustre and Esterel, the proposed model represents concurrency with the concept of process; it can be seen as a third point of view on interactive systems, more adapted to the need of user interfaces and making interaction its core concept. In contrast with them, its design integrates software engineering concerns and particularly the need of strong conceptual unification so as to support flexible development processes. Moreover, our work extends the approach followed by SCADE Display concerning the verification capabilities of interactive applications: when SCADE adresses low-level related properties (related to the concrete user interface: graphics, code design) our approach also encompasses upstream user interface design phases.

THE DJNN FRAMEWORK

`djnn` (available at `http://djnn.net`) is a programming framework that relies on a model of interactive software in which any program can be described as a tree of interactive components [7]. Basic components such as variables, control structures, and graphical objects are assembled to produce bigger components, themselves assembled until producing the desired application.

The execution of a program is described by the interactions between its components, and between them and the external environment: components react to events detected in their environment, and may themselves trigger events. For instance, a simple "fire alarm" program can be described with three components. The first is activated when the temperature is higher than a threshold, the second produces a sound when activated and the third binds the two others by propagating the activation of the first one to the second one.

Such a component model applies to input and output devices. This allows djnn to provide support for a wide range of devices, thus fostering the exploration of wide design spaces. But djnn has also the expressive potential of a general programming language. This contrasts with most user interface programming frameworks, which provide reusable components and architecture patterns that programmers combine with code written in a traditional programming language. Not only does djnn aim at covering 100% of the user interface code, it also has the potential of describing the functional core as well, thus covering whole interactive applications.

Theoretical foundation: interactive processes

Like functional programming languages, the conceptual model of djnn relies on a very reduced set of basic concepts from which all other language concepts and programmer-defined concepts are derived. In functional languages the basic concepts are functions, arguments and function calls, all rooted in the theoretical concept of lambda term from lambda calculus. In djnn the basic concepts are components, names and activation, all rooted in the theoretical concept of process from process algebras [3].

While computations can be described as the evaluation of lambda expressions, interactions can be described as the activation of interconnected processes: activation signals, called events, propagate from one process to another according to how they are coupled, thus producing the reactive behavior of the system. This concept of interaction differs from that of communication used in most process algebras, but it is nevertheless possible to express formal models of component activation in existing process calculi.

Process-based theories are general enough to model both interaction-oriented software and computation-oriented software [13]. But they can also model hardware devices, and more generally the environment in which software applications run. This allows to model both the software and its direct environment using processes, so that no special provision has to be made for those part of the software that interact with the environment and the user. Whole interactive applications can therefore be described in the same language.

Core interactive components

Components are man-made embodiment of processes: engineers build systems by assembling components, and the theoretical model of the resulting systems can be deduced from those of the individual components and how they are assembled. Programmers create interactive programs by instantiating and assembling software components, and connecting them to hardware components. The djnn environment provides them with basic software components to this effect:

- input components represent input devices, sensors, or elements of the execution context. Their activation is coupled to external events. For instance, a mouse is an input component made of smaller input components such as a buttons and a position tracker. The position tracker itself has two smaller input components named X and Y.
- output components represent output devices or abstractions used to manipulate output devices. Their activation is coupled to actions performed by the output devices. For example, a graphical object is an output component and any activation of it or of its sub-components triggers changes in the display.
- data components represent the computer memory. The smaller data components are named properties, and correspond to the basic types. Properties have sub-components named READ and WRITE, but like in traditional languages the usual practice is to reserve their use to specific components such as operations as assignments. The activation of the WRITE sub-component is the basis of dataflow: modifying a value can be used to trigger actions such as copying the value to another.
- operation components represent the operations provided by the execution platform, usually the CPU.
- control structure components represent the various ways in which the activation of components can be controlled. A control structure is a component that creates couplings between other components when it is activated. Control structures are the key to supporting the diversity of control patterns used in interactive software. Standard control structures range from the binding, which creates a simple coupling when it is activated, or the connector, which creates a coupling between a source and a copy instruction, to finite state machines and Statecharts. Other control structures can be created at will by combining existing components.

Assembling components

Complete applications and reusable components can be created from the basic components described above. Basic components are assembled to create large components that implement small interactors. These interactors can be assembled to create larger interactors, and so on.

A simple button, for example, can be built from few graphical shapes (rectangles, gradient, color, text, etc.) associated with a switch and a finite state machine that control the appearance of the button on mouse or touch events.

```
component button (posX, posY, label) {
  component click
  switch sw {
    component released {
      color c1 (200, 200, 200)
    }
    component pressed {
      color c2 (100, 100, 100)
     }
  }
  rectangle rec (posX, posY, 60, 30)
  text t (posX + 15, posY + 15,  label)
  FSM fsm {
    state released
    state pressed
    transition (released, press, rec, "press", 0)
    transition (press, released, rec, "release", click)
  }
  connector (fsm, "state", sw, "state")
}
```

The button can then be inserted in a more complex component to trigger a specific process through a simple binding.

```
component alarm {
  button b1 (50, 50, "alarm")
  beep fire_alarm
  binding (b1, "click", fire_alarm, "start")
}
```

Verifying properties

The djnn framework gives a central role to the tree structure of programs. In particular, mimicking graphical scene graphs introduced a few decades ago, the tree structure is used to express execution control. As a consequence it becomes possible to evaluate some properties by a static analysis of the tree through pattern matching techniques such as XPath requests. For example, the position of a graphical object in the tree tells its relative position to other graphical objects in the same component. Thus, a component situated on the right of another one in the tree will be displayed on top of it if their coordinates overlap. In the same way, djnn implements a flavor of graphical scene graphs in which graphical style components such as color, opacity and stroke width can be placed in the tree and act as context modifiers that affect all the shapes that follow see Figure 1).

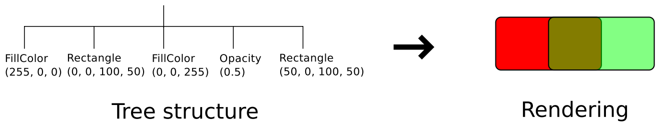


Figure 1. The order in the tree determines the graphical result

Pattern matching over the tree structure also enables the verification of component signature. This is a strong requirement in the context of the parallel development of complex graphical user interfaces where it is needed to ensure that a component respect a contract. Suppose for example that one creates a new widget for a WIMP toolkit. It must then be checked that this widget has a `width` and a `height` children to ensure that it will be possible to connect them to the layout system. Such a verification can be made by checking that the XPath expressions `expr=(/widget/width)` and `expr=(/widget/height)` over the component do not return a null result.

Finally, the combination of Xpath queries to select elements and simple algorithms over their results allows to address more complex case such as the verification of the control flow connecting a sensor to an alarm through various computation units [8].

The djnn platform

djnn is currently available as a collection of libraries and as an interpreter. The core djnn library manages the execution model and the component system. The other libraries each bring a collection of components dedicated to an interaction modality: graphics, input, gesture, sound, etc. Programmers can use these libraries like they would use any programming framework, by writing programs that use the djnn API to create components. The main difference with programming frameworks is that they can create entire applications with component-creation instructions only.

The core djnn library also implements parsers for external component formats, currently XML and Json. Therefore, it can be turned into a component interpretor that reads components from files and executes them. Components can be stored in multiple files in order to support various software engineering processes. In particular, user interface design teams like to store their graphical components in separate files, so that they can be managed independently by graphical designers and merged with the rest of the application at run time only.

APPLICATION SCENARIOS

As an example development process based on djnn, consider an aircraft manufacturer who wants to add new interaction functionalities on a particular cockpit subsystem. This translates into various engineering phases at growing technology readiness levels, from initial exploration to final system development and validation. At lower TRLs, the design processes will ideally reflect the state of the art of user interface design. At higher TRLs, they must reflect the state of the art of requirements engineering and code validation.

In this section, we illustrate a proof of concept experiment where, after observing current practices and gathering requirements from various actors of the aeronautical industry, we tested the role that djnn could play in supporting work at both lower and higher TRLs. The proposed scenarios are very simplified and exaggerated compared to the original industrial situations. Nevertheless, they serve well to illustrate the benefits that could be expected from a unified architecture and execution model such as djnn's.

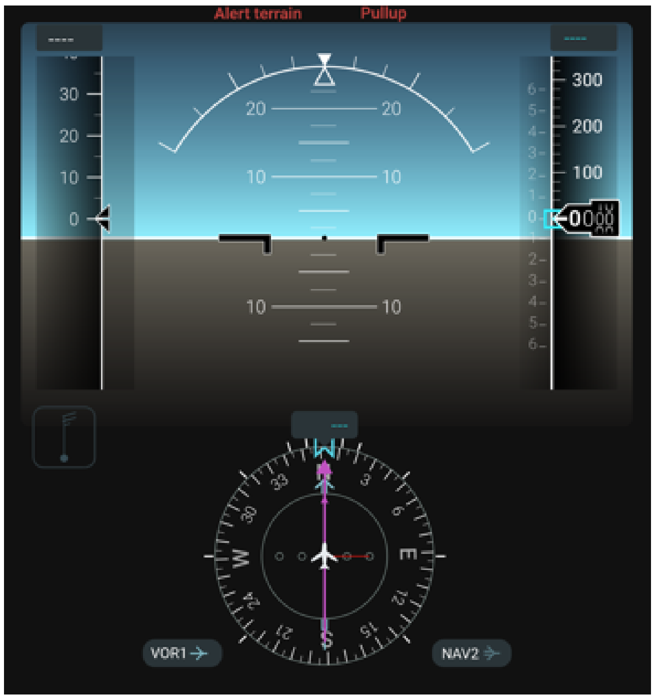


Figure 2. A primary flight display

We focus here on the design process of a primary flight display (PFD) i.e. the instrument that displays the basic parameters of the plane. The PFD consists of six parts, as shown in Figure 2:

- Attitude indicator: also known as artificial horizon, it gives information about the pitch (fore and aft tilt) and roll (side to side tilt) of the aircraft (center of the PFD)
- Altitude indicator (right side).
- Airspeed indicator (left side).
- Heading display: displays the magnetic heading of the aircraft (bottom).
- Source selector buttons: control the display of radio magnetic indicators on the heading display (bottom).
- Alerts: alert messages show at the top of the display.

In this prototype, the attitude component has an additional interactive part. Besides displaying altitude of the plane, this component gives an indication on the target altitude for an autopilot. However, while in traditional cockpits, the pilot sets the target altitude with a physical button in a distant part of the cockpit, here the team wants to explore the direct manipulation on the PFD.

In the following scenarios, a group of designers work with test pilots to build a prototype of the graphical appearance and the behavior of a new PFD, then the prototype is sent to the equipment provider. There, a team of developer codes the component and runs automated checks to verify that it has the same behavior as the prototype, while another team runs checks to verify properties that the component needs for inclusion in the existing code-base. Later, the component is returned to the manufacturer, who can run automated checks to verify its conformance with the original prototype. Through the examination of the design process of two components, we show here how the `djnn` framework supports concurrent development and enables the verification and validation of components.

Increasing fidelity

In the cockpit, the PFD is part of a more complex system of sensors and physical interactors. The team needs to simulate these subsystems in order to test and refine the behavior of the attitude component. At low TRL, the sensors do not have to be realistic. Therefore, the developer chooses to connect the attitude indicator to the best compatible sensor at hand: the motion sensor of his laptop computer. This allows him to perform very fast development iterations. When he is satisfied with his work, the developer needs to send the component to the lead developer who is in charge of integrating several components. He dumps the component to XML format and sends the resulting file to her. The lead developer just has to drag and drop the XML file to her project directory, where it will sit with the other components she has received. Running the master PFD component will load all the components from the directory and execute them. She can use her own motion sensor for testing the result. When the component is mature enough, it can undergo a review process to migrate to a higher TRL. The team in charge of this must test the component against a more realistic environment: a simulation engine controlled by a joystick. They save the component to a directory on the test machine with the proper equipment. The component needs to be adapted to the joystick used for pitch and roll. If necessary, this can be done with a simple rewiring; no modification is required in the component itself. But here, the original developer has added an adaptive behavior to the component: when a joystick is detected, the rewiring is performed automatically, and the component can be used as is.

Concurrent development

A programmer and a graphical designer are producing a component that displays the heading of the plane. Using the initial specification of the component structure as a contract, they can work in parallel. While the designer is creating the graphical skin and layout of the component, the programmer implements the behavior. Here, she uses temporary graphics that the graphical designer has sent earlier (Figure 3a).

When the graphical designer produces graphics, he saves them in SVG format, an XML-based markup language supported by all major vector graphics authoring tools. In this form, the graphics are considered as `djnn` components and can be manipulated like any other component. The developer can add them to the component he is working on, address them by their name, and connect them to other subcomponents. The names of graphical components in the SVG file are the implementation of the contract between the graphical designer and the developer.

When the final graphics are ready (Figure 3b) the graphical designer can send them to the developer. Replacing the temporary graphics with the new ones is as simple as replacing the SVG file in the appropriate directory and restarting the component. In our case, the graphical designer is late and the component has already been sent to the project manager who gives a demo to visitors in a few minutes. This poses no problem: the project manager saves the file, restarts the component, and the demo is ready.

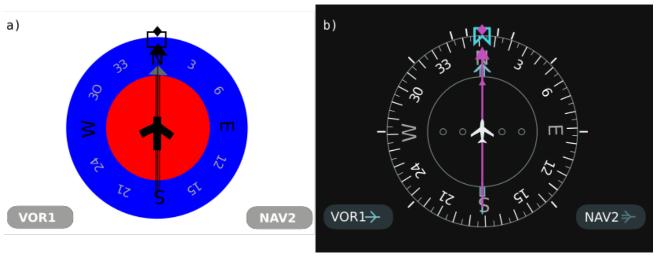


Figure 3. Heading display: (a) sketch (b) final graphics

Parallel design

The altitude component, besides displaying the altitude of the plane, also gives an indication on the target altitude for an autopilot. In traditional cockpits, the pilot sets the target altitude with a physical button in a distant part of the cockpit. For this new design, the team wants to explore the direct manipulation on the PFD. During design sessions, the team has identified several interaction variants. One prototype is developed in order to explore them further with final users. The first option relies on sliding the element that represents the target altitude (the blue element in Figure 4a). The second option is an indirect interaction on a smartphone-like number picker (Figure 4b). Switching from one option to the other is just a matter of loading one component or the other. Here, usability tests highlight manipulation problems with the first option and pilots validate the second one.

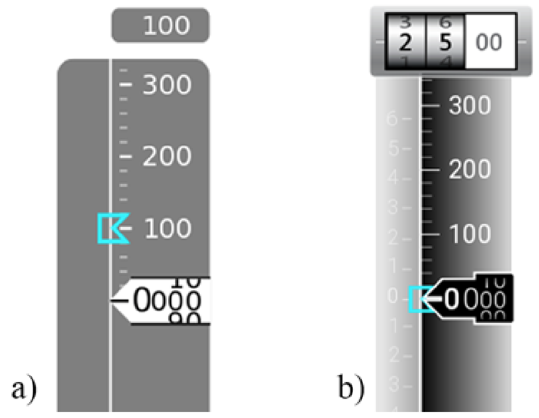


Figure 4. Altitude indicators: (a) direct manipulation and (b) indirect interaction for setting the target altitude

Component verification and validation

The airspeed component is subcontracted to an external provider, who delivers it through a web site. The project manager can either download the component and copy it in the component directory as described above or have it loaded dynamically by referencing the URL in the program. He needs to verify that the component will work well with the rest of the application. To do so he runs an automatic verification against the specification of the component, made of a collection of properties. This verification is made on the XML form of the `djnn` component, using pattern-matching techniques. In the first version of the component, the "ground speed" property is missing and the verification process automatically notifies it. After notifying the subcontractor, the verification is successful for the revised version received, and the component is loaded.

After the integration of all sub-components to the PFD component, the lead developer verifies that all the control flows are well wired. For example, she needs to be sure that alarms will be displayed when required. She uses a program that checks the control flow. The program traces back the control chain from "alert terrain" up to the "altitude" property and signals that it is not connected to any value. In the mean time, the verification raises a warning on the attitude component: two dataflow chains are connected to the same value. Thanks to these warnings, she can deduce than the simulated value of the altitude is connected by mistake to the attitude property, correct it and be certain that the alert terrain alarm will be displayed when needed.

CONCLUSION AND PERSPECTIVES

In this article, we proposed a general framework that provides a semantical unification of control mechanisms in interactive software. This framework relies on a framework whose basic elements are processes, names and events, from which more complex control structures can be defined (transfer of control, activation, interruption, transfer of data, deterministic choice, state machines, etc). On the top of these, a large collection of components have been designed for implementing the pragmatic aspects interaction: management of display (graphics, text, color, ...), input management (mouse, multi)touch), sound, interface with various external devices, etc. `djnn` is the name of the proposed implementation of this framework. It provides interaction designers and software developers with a platform for developing new components (through a dedicated language), assembling them and running the resulting application.

One long term objective for `djnn` is to provide the industry with a tool for development of interactive applications adapted to their domains. For this purpose, two major axes must be studied: for the moment, `djnn` is not supported by an IDE (Integrated Development Environment) similar to Codeblock, Eclipse or TopCased. Such tools useful for `djnn` users must be developed, especially a graphical editor for `djnn` models. The question of certification also has to be tackled. In the context of aeronautic, if `djnn` is used to develop and verify aircraft on-board applications, it must comply with some normative requirements related to software tool qualification (as specified in [1]).

The framework encompasses mechanisms dedicated to the expression and the verification of properties specific to interactive applications. Based on abstract interpretation, this allows to directly check on the code various properties at design- or system-level. Abstractions based on the component graph as well as the control flow graph retrieved from the code allow to address properties related not only to low level abstraction (code) but also to higher level (user interaction). As all necessary information describing the interaction is available at code level, our approach does not suffer from classical impossibilities related to absence of high level information for verification. Even if promising results for verification have already been obtained, a lot of research remain to be done to explore all the possibilities enabled by this approach.

ACKNOWLEDGEMENTS

This work was supported by DGAC projects Fenics and Fumseck, and the EU ARTEMIS JU through project HoliDes (http://www.holides.eu/) SP-8, GA No.: 332933. Any contents herein reflect only the authors' views. No supporting agency is liable for any use that may be made of the information contained herein.

REFERENCES

1. DO330. Software Tool Qualification Considerations, 2012. RTCA,Inc.

2. Aït Ameur, Y., Baron, M., Kamel, N., and Mota, J.-M. Encoding a process algebra using the Event B method. *STTT 11*, 3 (2009), 239–253.

3. Baeten, J. C. M. A brief history of process algebra. *Theor. Comput. Sci. 335*, 2-3 (May 2005), 131–146.

4. Berry, G., Bouali, A., Fornari, X., Ledinot, E., Nassor, E., and De Simone, R. ESTEREL: A formal method applied to avionic software development. *Science of Computer Programming 36*, 1 (Dec. 2000), 5–25.

5. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Mine, A., Monniaux, D., and Rival, X. *Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software*. LNCS 2566. Springer, Oct. 2002, 85–108.

6. Chatty, S. Extending a graphical toolkit for two-handed interaction. In *Proceedings of the ACM UIST*, Addison-Wesley (Nov. 1994), 195–204.

7. Chatty, S. Supporting multidisciplinary software composition for interactive applications. In *Proc. of the 7th international symposium on software composition*, no. 4954 in LNCS, Springer Verlag (2008), 173–189.

8. Chatty, S., Magnaudet, M., and Prun, D. Verification of properties of interactive components from their executable code. In *Proc. ACM EICS'15*, ACM (2015).

9. Cousot, P., and Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. ACM POPL'77*, ACM (1977), 238–252.

10. Duke, D. J., and Harrison, M. D. Abstract Interaction Objects. *Computer Graphics Forum 12*, 3 (1993), 25–36.

11. Elliott, C., and Hudak, P. Functional reactive animation. In *International Conference on Functional Programming* (1997), 263–273.

12. Esterel Technologies. Scade display HMI software design. `http://www.esterel-technologies.com/products/scade-display/`.

13. Goldin, D., Smolka, S., and Wegner, P., Eds. *Interactive computation - the new paradigm*. Springer-Verlag, 2006.

14. González-Calleros, J. M., Guerrero Garcia, J., and Vanderdonckt, J. Advanced human-machine interface automatic evaluation. *Universal Access in the Information Society 12*, 4 (2013), 387–401.

15. Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D. The synchronous dataflow programming language Lustre. In *Proceedings of IEEE*, no. 9 in 79 (September 1991), 1305–1320.

16. Hussey, A., and Carrington, D. *Specifying a Web Browser Interface Using Object-Z*. Springer, 1998, 157–174.

17. Le Charlier, B. Abstract interpretation and application to interactive system verification. In *Proc. DSV-IS'96*, Springer (1996), 46–72.

18. Myers, B. Separating application code from toolkits: Eliminating the spaghetti of callbacks. In *Proceedings of the ACM UIST*, Addison-Wesley (1991), 211–220.

19. Myers, B. A., and Rosson, M. B. Survey on user interface programming. In *Proceedings of ACM CHI'92*, ACM (1992), 195–202.

20. Palanque, P., Barboni, E., Martinie, C., Navarre, D., and Winckler, M. A model-based approach for supporting engineering usability evaluation of interaction techniques. In *Proc. EICS 2011*, ACM (2011), 21–30.

21. Thimbleby, H., and Gow, J. Applying graph theory to interaction design. In *Proc. EICS2007*, J. Gulliksen, Ed., vol. 4940 of *LNCS*, Springer Verlag (2008), 501–518.

Session 29

Multicore & ARM

Friday 29th, 11:45 – Guillaumet

Felix Baum **felix_baum@mentor.com**
Mentor Embedded

Making Full use of Emerging ARM- based Heterogeneous Multicore SoCs

Abstract

The complexity and pace of heterogeneous SoC architectures is accelerating at blinding speed. While these complex hardware architectures enable product vision, they also create new and difficult challenges for the system architect. Running and debugging an Operating System and application code on a single core is child's play. This is also true for running Synchronous Multiprocessing (SMP) capable Operating Systems on homogenous multicore processors.

The modern day SoC combines asymmetric multiple cores, graphics processing units, offload engines and more on a single piece of silicon. This paper will discuss opportunities for system partitioning and consolidation, and some of the key issues and challenges of architecting, developing and debugging software on these complex systems.

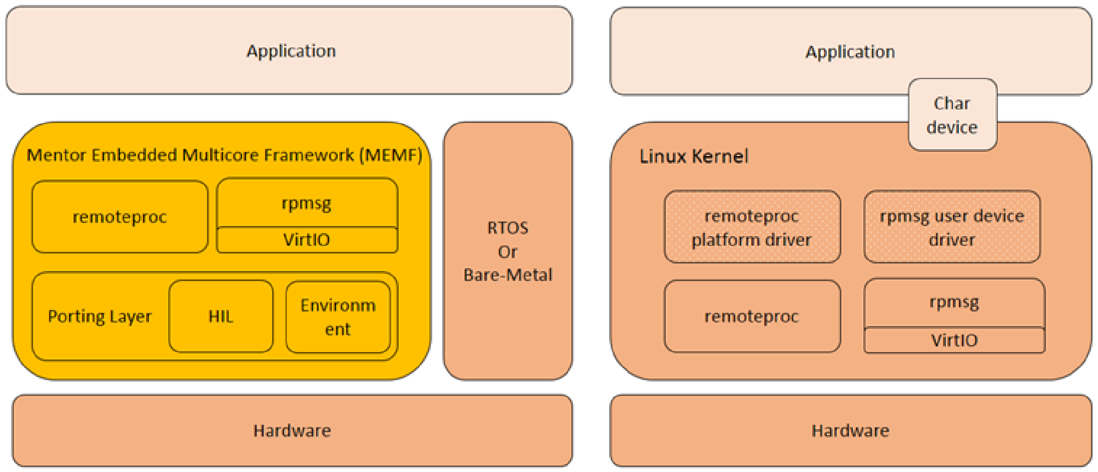
Introduction

Heterogeneous multicore systems combining two or more different types of microprocessors (MPUs) and microcontrollers (MCUs) are quickly becoming the de-facto architecture in the embedded industry today. The quick emergence of these systems can be attributed to a number of factors, with one of the main one being consolidation. Over the last few years, there have been explosion of adoption in embedded designs of ARM cores. One of the benefits of using ARM is the ease with which SoC hardware designers can efficiently solve computing problems using systems of heterogeneous cores. Designers are able to allocate the right amount of compute for a given problem at the point that compute power is needed. And while SMP operating systems provide the infrastructure required to balance application workload symmetrically or asymmetrically across multiple homogeneous cores present in a multiprocessing system, in order to leverage the compute bandwidth provided by the heterogeneous processors present in the system, AMP software architectures should be employed.

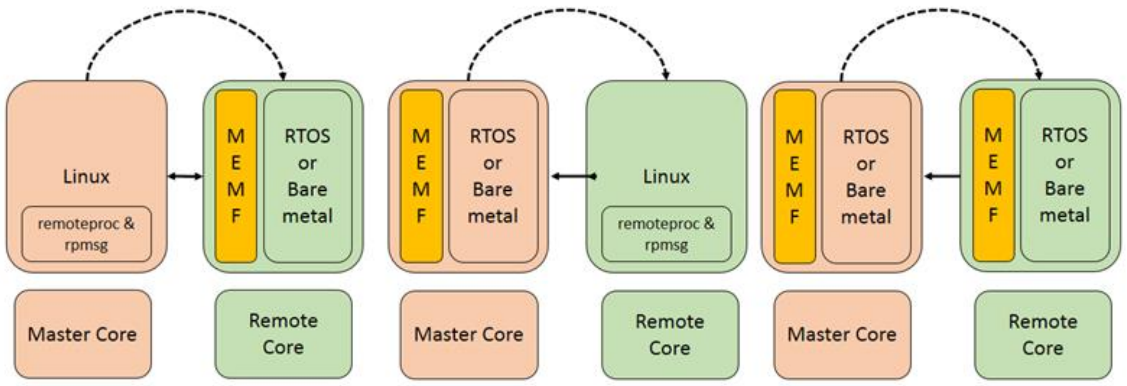
AMP architectures typically entail a combination of dissimilar software environments such as Linux®, a real-time operating system (RTOS), or bare-metal running on homogeneous or heterogeneous processing cores present in the SoC – all working in concert to achieve the design goals of the end application. Typical designs involve a software context on a master core bringing up a remote software context on a remote core on a demand-driven basis to offload computation. The master, remote processors, and their associated software contexts (i.e., OS environments) could be homogeneous or heterogeneous in nature. In order to effectively deal with the complexities of managing life cycle of several different operating systems on possibly dissimilar processors, and to provide an enabling Inter Processor Communications (IPC) infrastructure for offloading compute workload, new and improved software capabilities and methods are required.

To ensure that developers can efficiently solve compute problems with heterogeneous ARM SoCs, it is important to standardize some of the frameworks used to engineer various aspects of heterogeneous ARM systems. The Multicore Framework described in the paper is a software framework that provides two key capabilities to AMP system developers: 1) It provides the remoteproc component and API for life cycle management of remote processors and their associated software contexts; and 2) It provides rpmsg component and API for Inter-Processor Communications (IPC) between OS contexts in the AMP environment. The Framework hides the complexities of managing heterogeneous hardware and software environments providing a simplified application level interface to the user.

Compliance to open standards and adoption by the open source Linux community are important considerations when choosing an appropriate API for the multicore framework outlined in the paper. With these considerations, we choose the remoteproc and rpmsg API present in the Linux 3.4.x kernel and newer. The Linux remoteproc and rpmsg infrastructure was originally conceived and committed to the Linux kernel by Texas Instruments. The infrastructure allows Linux OS on a master processor to manage life cycle and communications with remote software context on a remote processor. However, the Linux provided infrastructure has some caveats; Linux rpmsg implicitly assumes that Linux will always be the master operating system and does not support Linux as remote OS in an AMP configuration. Further, the remoteproc and rpmsg APIs are available from Linux kernel space only – there is no equivalent API or library usable with other OSs and run-times. We developed a standalone library written in C language that provides a clean room implementation of the remoteproc and rpmsg functionality usable with RTOS or Bare-metal software environments, with API level compatibility and functional symmetry to its Linux counterpart.



In the framework we are providing workflows to package Linux as a remote OS, and the required run-time infrastructure to boot Linux as remote OS in AMP configurations. And some of the configurations outlined in the paper are provided on this diagram:



The framework finds direct applications and is well suited for un-supervised Asymmetric Multi-processing (AMP) architectures - where the participating operating systems run natively on the processors present in the system. It provides a simple and effective architecture allowing application developers leverage available compute resources without the complexities and overhead associated with supervised AMP architectures that involve a Hypervisor. Of course for applications that require isolation, and virtualization between the OS environments, a Hypervisor based supervised AMP architecture is the best fit.

In supervised architectures, the participating guest operating systems run on virtual machines managed and scheduled by a Hypervisor which provides isolation, and virtualization functions for the guest virtual machines. This framework finds application in supervised architectures as well. It can be used from either the guest OS context for un-supervised management of heterogeneous compute resources present in the system. Alternatively, it can be used from within the hypervisor for supervised management of heterogeneous compute resources, allowing the hypervisor to supervise interactions between guest operating systems and remote contexts on heterogeneous cores.

The multicore framework finds applications in various scenarios. In general, it is well suited in situations requiring off-load of compute functions to specialized processing cores present on a multi-processing chip. In most cases, when the remote compute resource is not needed it can be kept in a low-power standby state, and can be brought up on a demand-driven basis allowing for optimal power usage in power constrained applications. It promotes and simplifies for consolidation of discrete embedded systems onto a single multi-processing SoC. It enables an easy migration path for legacy software on legacy uni-core silicon to easily inter-operate with enhanced system functionality developed on newer and more powerful multi-processing chips. It also enables fault tolerant architectures; a Safety certified RTOS handling critical system functionality can manage a Linux context handling non-critical functions using MEMF. On failure of Linux based sub-system, the RTOS can simply re-boot Linux using the features of the framework.

Development of AMP systems poses a unique set of challenges. System developers typically find themselves in situations having to simultaneously debug several different OS environments deployed on dissimilar processors on heterogeneous SOCs. Having a unified debugging environment with awareness of operating systems involved will enhance the debug experience and improve productivity. We will point out and describe debug tools that provide a unified IDE with OS awareness for all OS environments and supports a multitude of debug options; JTAG based debug for Linux kernel space, RTOS and bare-metal contexts, GDB based debug for Linux user space, and RTOS based applications.

Software profiling could be a valuable tool to gain insight into how various applications deployed on heterogeneous operating systems interact with each other system during an AMP system's run-time. Each OS instance is typically based of an independent clock source, and any profiling data collected within a given OS instance will be based on a time base that is local to the OS. Graphically visualizing and analyzing trace data collected from disparate OS sources in a unified time reference is a very powerful tool and allows users to gain interesting insights into complex interactions occurring in the AMP system.

Summary

The initial implementation of the Multicore Framework described in this paper was open-sourced under the OpenAMP open-source project with support for the Zynq 7000 SOC. OpenAMP as an emerging API standard managed under the umbrella of Multicore Association. This project is jointly maintained by

Mentor Graphics and Xilinx. A current reference implementation of the proposed OpenAMP standard is available at: https://github.com/OpenAMP/open-amp. Mentor Embedded Multicore Framework (MEMF) is a proprietary implementation of the OpenAMP standard.

MEMF is tightly integrated with and readily supported by all Mentor provided OS run-times. It supports a diverse set of ARM based SOCs and platforms. Using MEMF with Mentor's tools and operating systems obviates users from having to design their AMP system from scratch. i.e., perform tasks discussed under the System level considerations section. Users can focus on AMP application development with one of the pre-canned reference configurations and later customize the system configuration to fit their needs.

Keywords

Multicore SoC, Synchronous Multiprocessing, Asynchronous Multiprocessing, AMP, SMP, ARM, Mentor Embedded Multicore Framework, MEMF, OpenAMP

Session 30

Worse Case Execution Time

Friday 29th, 11:45 – Ariane 1

On the Reliability of the Probabilistic Worst-Case Execution Time Estimates

Fabrice Guet, Luca Santinelli, and Jérôme Morio
ONERA - The French Aerospace Lab, Toulouse France
{fabrice.guet|luca.santinelli|jerome.morio}@onera.fr

1. INTRODUCTION

The execution of software tasks within real-time systems needs to be analysed with respect to both functional and non-functional constraints. In particular, real-time systems require strict timing evaluations of the tasks execution behavior, especially their Worst-Case Execution Time (WCET).

Safety-critical embedded systems exhibit execution time variability, although classical real-time modeling and analyses account only for the worst-case. The systemic complexity of real-time systems comes from the hardware complexity (e.g., current multi-core architectures, shared resources such as memory, and speculative mechanisms like cache memories and pipelines [13, 23]), the software complexity (e.g., multiple embedded functionalities, wide interoperability, co-existence of functional and non-functional constraints), complex system component interactions and dependences, and diverse environments. All of them participate to the variability in the temporal behavior of the tasks.

Regarding this systemic complexity, probabilistic approaches are emerging as effective alternative to deterministic approaches for WCETs estimate. Their objective is to characterize system execution variabilities with probability distributions that associate to multiple possible WCETs their probability of occurence within a system execution trace, on contrary to deterministic approaches that provide a single WCET estimate. The challenge is to ensure the predictability based on the probabilities. So far, the probabilistic approaches are less costly in modeling task execution behavior and more accuracy with regard real-time systems average performances compared to the deterministic approaches.

This paper focuses on Measurement-Based approaches for Timing Analyses (MBPTA). MBPTA relies on both execution time measurements and the application of the Extreme Value Theory (EVT). Thus an exact model of both hardware and software is not required, contrary to deterministic approaches, as the measurement of the actual system behavior is sufficient. The MBPTA provides probabilistic Worst-Case Execution Time (pWCET) estimates[1] [6, 18]. Currently, the main problem of the MBPTA is the lack of mathematical robustness since EVT actual application relies on non systematic statistical approaches.

Hardware systemic effects in real-time systems [33] make EVT applicability difficult with regard to its required theoretical hypotheses. It is necessary then to ensure the applicability of the EVT to realistic embedded systems (non time-randomized embedded systems). Moreover, real-time systems require strong guarantees on the pWCET estimates thus, diagnostic tests have to be introduced to check hypotheses for generalizing the EVT applicability to any embedded systems [37].

In this paper we propose the logical workflow that checks the applicability of the EVT for the pWCET estimation problem. The proposed framework is a DIAGnostic tool for the eXTReMe value theory, named diagXtrm. The tool applies tests and makes a decision on the reliability of the resulting pWCET estimate without human intervention. The objective is to establish a systematic and reproducible process for estimating the pWCET which is able to cope with both performance and safety of existing as well as future real-time systems.

Organization of the paper. In Section 2 we relate the WCET problem, especially for the MBPTA, by depicting existing approaches and stressing the novelty of the proposed framework. In Section 3, we set the basics of the real-time probabilistic modeling and focus on the theoretical aspects of the EVT applicability. Section 4 presents the main steps of the diagXtrm tool, and Section 5 develops the tests that compose the tool. In Section 6 diagXtrm is applied to a realistic hardware platform running a set of tasks. Section 7 is for conclusions and future work.

2. RELATED WORK

Estimating the WCET of a task for hard real-time systems has been addressed in many ways [40]; all differ depending on the kind of hardware architecture.

Platforms are said to be deterministic whenever the execution time of a task is the same for the same input data. They are said to be non deterministic instead, whenever the execution time varies for the same input data. The non determinism comes from hardware components like cache memories, pipelines, etc. [35].

Static deterministic timing analysis and measurement-based deterministic timing analysis are effective for deterministic platforms. Static approaches provide safe WCET estimates as they are proved to be the worst. They rely either on an exact modeling of the system and a complete exploration of all its state or on a simplified version where some conditions are respected or even enforced. Measurement-based approaches provide timing behavior upper-bounds as distributions that overcome most of the possibilities. That is the reason why static approaches are preferred on measurement-based ones to give high guarantees on the system constraints. Nevertheless, when it comes to non deterministic architectures, static approaches produce pessimistic WCET estimates due to the overall systemic complexity; the analytical modeling phase is more and more difficult, the models confidence decreases, and the resulting estimates deteriorate [7]. However, tools based on the static modeling of both hardware and software aspects

[1] pWCETs are alternative to deterministic Worst-Case Execution Times as distributions with multiple extreme execution times, each with a probability of happening.

are able to provide safe but pessimistic WCET estimates because they take into account the worst-case at every modeling step. The estimates could be far from actual measurements and hardware performances.

The non determinism resulting from enhanced performance and the consequent execution time variability question the deterministic approaches. Facing this new challenge, probabilistic approaches tend to emerge: they can be either Static-based [18, 9] or Measurement-Based Timing Analyses (MBPTA). diagXtrm is a MBPTA approach and is able to capture well the systemic effects together with the coherence mechanisms between shared resources. As it relies on end-to-end measurements of the task execution time, it does not require a huge amount of information or exact hardware nor software models. The probabilistic worst-case profiles are derived on the basis of the set of execution time measurements. Nevertheless, as MBPTA relies on measurements, the lack of completeness of experimental conditions can lead to unsafe pWCET estimates due to unobserved execution conditions.

2.1 MBPTA approaches

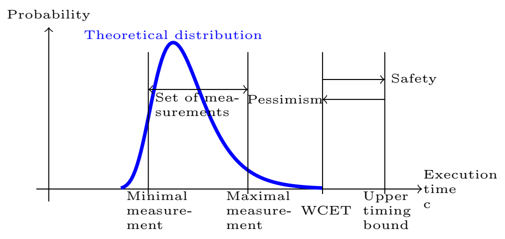


Figure 1: Overview of the WCET problem. Example of a timing probabilistic profile of a task.

The objective of MBPTA approaches is to derive probabilistic profiling of the timing behavior of a task, like in Figure 1, through a statistical modeling. Such a profile has to tend to the true theoretical distribution of execution times. In particular, MBPTA approaches are interested in modeling extreme execution times, for characterizing the worst-case, i.e. the values far from the average execution times, and potentially not measured. The probabilistic theory that focuses on extreme values and large deviations from the average values is the Extreme Value Theory (EVT) [20].

The EVT is a probabilistic paradigm that aims at predicting the improbable, i.e. it enables to derive the probability of rare events without requiring too many simulations. The EVT for estimating the WCET of a task in a scheduling analysis is first used in [19] where the Gumbel distribution is applied for modeling the distribution of execution times. A first algorithm for applying the EVT appeared in [25]. It extracts values from a sequence of execution time measurements according to the block maxima paradigm[2] and fits a Gumbel distribution to the measurements. Then the fitted distribution is compared to the measurements through a χ^2-test to confirm the model, otherwise the process is applied again for another number of extracted values.

The EVT applicability for embedded systems is first questioned in [33] and in particular about the statistical independence and the continuity of the execution time measurements. Two directions emerged from those questions: 1) the randomization of the hardware for solving the independence problem. Within the PROARTIS project first and then the PROXIMA project [1, 2], the EVT is applied to artificial (ad-hoc) random systems (Random Replacement policies in cache memories),[4, 15, 17]; 2) the elaboration of a methodology for guaranteeing the applicability of the EVT from the strong fundamentals of its mathematical hypotheses [5, 8, 31, 37] and derive reliable pWCET estimates from any real-time system (time-randomized as well as non time-randomized).

The approach proposed in the diagXtrm tool is a MBPTA approach and aims at solving the problem of the EVT applicability for real-time systems (both time-randomized and non time-randomized systems) by pursuing the works in [19, 25, 33, 37]. It represents the first structured and formal approach designed as a logical workflow that evaluates the EVT hypotheses for guaranteeing the MBPTA estimates. diagXtrm applies tests proved to be efficient for the considered analyses and an automatic parameter estimate process to provide pWCET estimates with an associated confidence for the EVT hypotheses.

3. PROBABILISTIC MODELING

The EVT relies on measurements of the system performance parameters, here the execution time of a task τ, for estimating extreme behaviors, where the worst-case should lie. The variability of the execution time of a task motivates its definition as a random variable[3], denoted by $\mathcal{C}$, which picks different possible values within the set[4] $\Omega \subseteq \mathbb{N}$, i.e. the distribution support of execution times the task τ can take to complete with a certain probability. The definition of a random variable stands for the uncertainty that lies on the uncertain systemic effects that occur in real-time systems. Each measurement C_i at discrete time instant i, is stored in a trace $\mathcal{T}$ such that $\forall i, \mathcal{T}(i) = C_i$. The length of $\mathcal{T}$ is denoted by n.

Three equivalent representations are used for $\mathcal{C}$, each is a probability distribution function: for all possible execution time $c \in \Omega$, it exists i) the cumulative distribution function (CDF) $F_{\mathcal{C}}(c) = P(\mathcal{C} \leq c)$, ii) the complementary cumulative distribution function (CCDF) $\overline{F}_{\mathcal{C}}(c) = P(\mathcal{C} > c) = 1 - F_{\mathcal{C}}(c)$, and iii) the probability mass function $f_{\mathcal{C}}(c) = P(\mathcal{C} = c)$ (for a continuous random variable it is $f_{\mathcal{C}}(c) = \frac{d}{dc} F_{\mathcal{C}}(c)$). The discrete random variable $\mathcal{C}$, based on the execution time measurements is said to be the Execution Time Profile[5] (ETP) of the task τ.

One key element about the pWCET relates to its theoretical existence: *the pWCET exists but cannot be observed* since it is the distribution of extreme execution times that are very hard to measure and potentially impossible to observe. To measure execution times with very low probability (e.g., 10^{-9}), it would require a large amount of simulations and well defined experimental conditions. Moreover, the worst-case conditions have to be guaranteed to be explored making such approach very costly in terms of time and exploration conditions. The lack of completeness of the experimental conditions cannot ensure the existence of pWCET estimates directly from ETPs.

[2]The sequence is divided into blocks of same size and only the maximum value of each block is retained.

[3]A random variable is a variable whose value is subject to variations due to chance, i.e. randomness, in a mathematical sense. Generalizing, also non-variable execution time could be represented as random variables, with only one value and the probability of happening equal to 1. Since execution times are from measurements, they results into empirical random variables.

[4]Execution time can assume only discrete values as multiple of the system clock.

[5]ETPs are discrete distributions since task execution times can only be a multiple of the system clock.

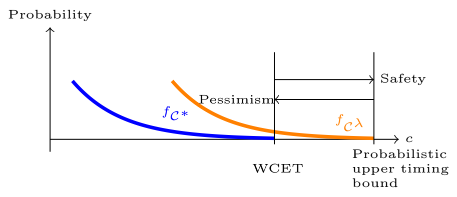


Figure 2: The pWCET estimate problem with the relationship between exact pWCET and pWCET estimate. An example of safe estimate $\mathcal{C}^\lambda$ with respect to the exact $\mathcal{C}^*$.

For formally defining the exact pWCET, we introduce the partial ordering of random variables by comparing their CCDF. Thus a random variable $\mathcal{C}_i$ is greater than or equal to a $\mathcal{C}_j$, $\mathcal{C}_i \succeq \mathcal{C}_j$, iff $P(\mathcal{C}_i > c) \geq P(\mathcal{C}_j > c)$, for every $c \in \Omega_{\mathcal{C}_i} \cap \Omega_{\mathcal{C}_j}$. Thus, the exact pWCET is the least upper random variable over all the ETPs for every execution condition. We denote the exact pWCET of a task by the random variable $\mathcal{C}^*$. Since exact pWCETs are impossible to obtain, as for any timing analysis approaches, we focus on pWCET estimates $\mathcal{C}^\lambda$. A pWCET estimate $\mathcal{C}^\lambda$ has to be safe i.e. has to be greater than $\mathcal{C}^*$, so $\mathcal{C}^\lambda \succeq \mathcal{C}^*$ like in Figure 2. The statistical modeling method from the EVT is the process we apply to achieve $\mathcal{C}^\lambda$.

3.1 Reliable pWCET estimates

The EVT is a widely used theory for predicting the improbable, i.e. giving probabilities of occurrence to extreme behaviors.

Under the hypothesis of *independent* and *identically distributed* (iid) execution time measurements $C_1, \ldots, C_n$ from an average discrete cumulative distribution function $F_{\mathcal{C}}$. The EVT ensures that the limit law of the maxima, i.e. the extreme execution times, denoted by $M_n = \max(C_1, \ldots, C_n)$ is a Generalized Extreme Value Distribution (GEV) H_ξ under norming constants such as the shape parameter $\xi \in \mathbb{R}$, the mean $\mu \in \mathbb{R} > 0$ and the variance $\sigma^2 \in \mathbb{R} > 0$ of the extreme execution times, with the Fisher-Tippett-Gnedenko theorem [20, 24].

This result implies that $F_{\mathcal{C}}$ belongs to the *Maximum Domain of Attraction* of the GEV H_ξ, denoted by $F_{\mathcal{C}} \in MDA(H_\xi)$. Given $\mathcal{C}$, whenever the iid hypothesis is respected and under good norming constants, the GEV is an appropriate distribution for the extreme execution times.

Depending on the value of ξ, the GEV can be either the Frechet ($\xi > 0$), the Gumbel ($\xi = 0$), or the Weibull ($\xi < 0$) distribution. In previous works the pWCET distribution has been assumed to be Gumbel, while here no assumption is made about the resulting GEV distribution and so there is no restriction on the values that ξ can take. The objective of the study is to get reliable pWCET estimates so that the distribution has to best-fit the measurements: the Gumbel can result from the best-fit or it can be imposed afterwards.

Considering $\mathcal{C}$ and $F_{\mathcal{C}}$, the CDF of the peaks $\mathcal{C} - u$ above the threshold u knowing $\mathcal{C} > u$ is

$$F^u(c) = P\{\mathcal{C} \leq u + c \mid \mathcal{C} > u\} = 1 - \frac{1 - F_{\mathcal{C}}(u + c)}{1 - F_{\mathcal{C}}(u)}. \quad (1)$$

If $F_{\mathcal{C}} \in MDA(H_\xi)$ then the limit law of the peaks is given by the Pickands theorem [36]:

Theorem 3.1 (Pickands theorem). *$F_{\mathcal{C}} \in MDA(H_\xi)$ iff*

$$\lim_{u \to c_0} \sup_{0 \leq c \leq c_0 - u} |F^u(c) - GPD_\xi(c)| = 0, \quad (2)$$

where c_0 is the potential WCET of τ. GPD_ξ the Generalized Pareto Distribution with the same shape parameter ξ as H_ξ, and F^u from Equation (1).

The Pickands Theorem states that for values above a threshold, the nearest the threshold is to the actual WCET (which is the task execution time right end-point for increasing values) the more the distribution of execution times tends to a Generalized Pareto Distribution.

Definition 3.2 (Generalized Pareto Distribution). *The distribution function GPD_ξ is the Generalized Pareto Distribution (GPD) defined as:*

$$GPD_\xi(c) = \begin{cases} 1 - (1 + \xi \times (c - u)/\alpha_u)^{-1/\xi} & if\ \xi \neq 0 \\ 1 - \exp(-(c - u)/\alpha_u) & if\ \xi = 0, \end{cases} \quad (3)$$

with $\alpha_u = \mu - \xi(u - \sigma^2)$, and defined on $\{c, 1 + \xi(c - u)/\alpha_u > 0\}$.

This fixes the basis of the EVT POT approach which consists in extracting the execution time measurements from $\mathcal{T}$ above a threshold u and fitting the experimental CDF with the continuous distribution function P_ξ. By applying the POT approach to the trace of execution times, the pWCET estimate which is the distribution of extreme execution times $\mathcal{C}^\lambda$ is a GPD.

For applying the EVT, one needs i) independent and ii) identically distributed execution time measurements from iii) a distribution in the Maximum Domain of Attraction of a GEV of shape parameter ξ. Those three elements are the hypotheses to check for having reliable pWCET estimates.

In practice, the independence hypothesis is difficult to assume because of history dependence in memory components as explained in Section 2.1. Moreover, the true distribution of the execution times is unknown and prevents from proving that execution times are identically distributed from a distribution in the Maximum Domain of Attraction of a GEV.

Further researches in the EVT domain proved the convergence of the Fisher-Tippett-Gnedenko theorem for stationary execution time measurements under two conditions [29, 30], and so the applicability of the EVT in the more general stationary case. The conditions especially relax the strict independence of the measurements and it is not necessary to know precisely the probabilistic law of the execution times as soon as they are stationary.

The strict hypotheses that prevented EVT applicability to non time-randomized embedded systems (todays systems), notably the independence, are so released allowing to apply the EVT to the pWCET estimate problem for any real-time system (both time-randomized and non time-randomized).

4. A DIAGNOSTIC TOOL FOR ESTIMATING THE PWCET WITH THE EXTREME VALUE THEORY

The main challenge of the MBPTA is the definition of a systematic approach that provides reliable pWCET estimates with the EVT. The reliability of a process comes from its definition: it is crucial to well identify the hypotheses and to choose both powerful tests and a proper parameter estimate process. A test is said to be powerful if it is able to reject a hypothesis when it is known to be false but also not reject it when it is known to be true. The reliability of the pWCET estimates holds if every hypothesis of the EVT is verified. Making use of the well defined tests and a proper estimate of the distribution parameters, here ξ and α_u, the reliability can be guaranteed.

The DIAGXTRM, by construction tends to reduce the sources of uncertainty that lie on the execution time measurements to fulfill the EVT hypotheses and the selection of the threshold [38]; moreover it quantify the estimates confidence. In that sense, the tool is a diagnostic of the stastical modeling with the EVT.

The tool is designed as a logical workflow which checks the applicability of the EVT with specific tests. For an input trace of execution times, DIAGXTRM provides a pWCET estimate $\mathcal{C}^\lambda$ and an associated confidence with regard to the EVT applicability hypotheses. The hypotheses to check on the trace of execution time measurements are: 1) stationarity, 2) short range dependence, 3) local independence of the peaks, 4) empirical peaks over the threshold follow a GPD. The four hypotheses define the hypothesis testing blocks includes in the main steps of the tool, described in Figure 3. In this section the DIAGXTRM is presented at a high level; the tests that compose it will be detailed in the following section.

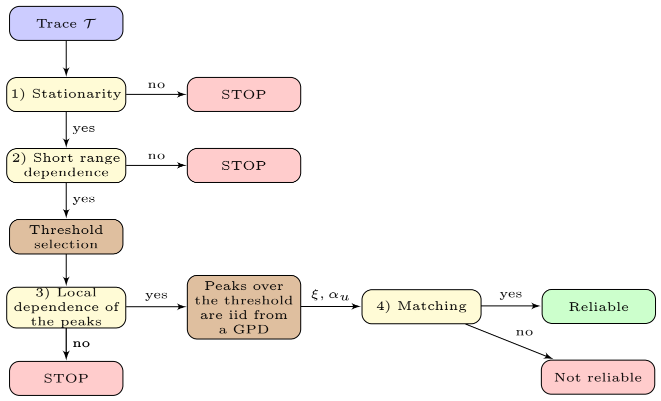


Figure 3: Decision diagram of the DIAGXTRM enlisting the tests and actions applied.

4.1 Design of the tests

DIAGXTRM is mainly based on the hypothesis testing theory that studies the rejection of a null hypothesis H_0. If H_0 is not rejected it is a necessary but not sufficient condition to satisfy H_0. The first step is to select an appropriate metric that evaluates the possibility of rejecting H_0, then the metric is applied to the trace $\mathcal{T}$ of execution time measurements returning a *result* through which making a decision about H_0.

In the design phase of the test, training sets are used to quantify the power of the metric for detecting H_0. The focus is on the conditional probability to reject H_0 knowing that H_0 is true $p-value = P(\overline{H_0}|H_0)$, which is the false positive rate of the test. The arbitrary threshold to reject H_0 is the value α defining the confidence interval for the test, hence for the hypothesis testing. A test may have a symmetric confidence interval, a two-sided test, otherwise this is a one-sided test. If the *result* of the applied metric to $\mathcal{T}$ is within the confidence interval, then H_0 is not rejected. Usually, α is chosen near 0 e.g., 0.01, 0.05 or 0.1, and corresponds to as many critical values *cvs* like in the two-sided test illustrated in Figure 4. If the *result* is in the darkest area, then H_0 is rejected but one has $\alpha \times 100\%$ of rejecting wrongly H_0.

We consider the possibility to fulfill H_0 [26], and use a fuzzy logic approach to test hypotheses. As the possibility to fulfill H_0 increases and so the confidence in H_0, the necessity to reject it decreases. Fuzzy logic is widely used to build decision making processes and is called robust statistics [12, 22, 26] when applied to statistics by quantifying the uncertainties associated to classical statistical approaches.

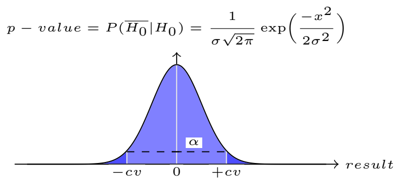

$p-value = P(\overline{H_0}|H_0) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-x^2}{2\sigma^2}\right)$
α
$-cv$
0
$+cv$
$result$

Figure 4: Hypothesis testing with a metric following a Gaussian law. (cv: the associated critical value to the α false positive rate)

For instance, instead of having or not a stationary trace of ETs, fuzzy logic quantifies whether the trace is near or far from the stationary model. The nearest it is the more confidence there is in the EVT applicability. Instead of one α level, 4 values are selected so that it is possible to either reject H_0 or accept H_0 with low, medium, high and full confidence level, corresponding to the $p-values$ 0.01, 0.025, 0.05 and 0.1. To resume, the approach we are formalizing for the pWCET estimation with EVT defuzzifies the statistical test by associating fuzzy $p-values$ to human-understandable confidence levels, depicted in Figure 5, and ease the decision making.

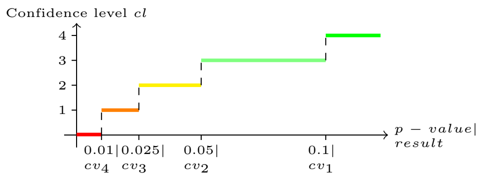


Figure 5: The DEFUZZIFICATION is a function from fuzzy $p-values$ (or equivalently their associated critical value (cv)) to confidence levels in $\{0;1;2;3;4\}$. For increasing confidence levels, H_0 is rejected or H_0 is accepted with low, medium, high and full confidence.

4.2 Decision making process

Each hypothesis testing block, blocks 1), 2), 3) and 4) in Figure 3, provides a *result* about the trace of execution times and so a confidence level as in Figure 5. Those confidence levels aims at reliable pWCET estimates with the EVT with regard to its hypothses applicability. One purpose of the fuzzy approach is to have a common scale for every test in order to aggregate each confidence level and to get a final confidence level on the pWCET estimate with the EVT. There exist many ways to aggregate the confidence levels, but one requirement is to have an aggregation in agreement with the tool specifications. In particular, the reliability is ensured when every hypothesis is guaranteed.

In the proposed process, there are four hypotheses to check: 1) the stationarity, 2) the short range dependence, 3) the local dependence of the peaks and 4) the matching with a GPD model. The final confidence level is denoted by $cl_{reliability}$ as a possibility metric to fulfill the whole process. Consequently, the confidence levels associated to each hypothesis are: cl_1, cl_2, cl_3 and cl_4. To statisfy the reliability requirement, if one confidence level is zero then the reliability has to be zero too. The confidence in the model is the barycenter of all the confidence levels so that it leads to Algorithm 1.

Algorithm 1 AGGREGATION algorithm of the confidence levels in the DIAGXTRM

```
1: $confidence\_levels \leftarrow [cl_1, cl_2, cl_3, cl_4]$  ▷ Previous analyses
2: procedure AGGREGATION($confidence\_levels$)
3:     if $\min(confidence\_levels) \geq 1$ then  ▷ Reliability
4:         $cl_{reliability} \leftarrow (cl_1 + cl_2 + cl_3 + cl_4)/4$  ▷ Reliability levels
5:     else
6:         $cl_{reliability} \leftarrow 0$
7:     end if
8: end procedure
```

Let H_0*: the EVT is applicable to* $\mathcal{T}$ be a null hypothesis, then $cl_{reliability}$ gives the confidence in fulfilling H_0. With regard to Algorithm 1, either H_0 is rejected for a null $cl_{reliability}$, or H_0 is accepted and in this case the higher $cl_{reliability}$ is, the more confidence in the model there is and thus in the pWCET estimates. The power of the tool to fulfill H_0 and to provide reliable pWCET estimates, depends also on the selected tests for each hypothesis. The DIAGXTRM is a high level methodology to provide reliable pWCET estimates, and one may easily replace a selected test in its respective hypothesis testing block by a better one thanks to new research works in time series (trace) analysis.

5. TESTS DETAILS

5.1 Stationarity analysis

Stationarity is an essential property in statistical analyses but it is usually assumed. The problem is even more diffcult because there is no systematic way to study stationarity and it often relies on subjective analyses [34].

DEFINITION 5.1 (STRICTLY STATIONARY TRACE). *A trace* $\mathcal{T} = (C_1, C_2, \ldots)$ *is a strictly stationary trace if for all* j, k, l*, the set of execution times* $C_j, \ldots, C_{j+k}$ *has the same probabilistic law as the set* $C_{l+j}, \ldots, C_{l+j+k}$.

If the execution time measurements in $\mathcal{T}$ are such that they respect Definition 5.1, then there is strong evidence that measurements are identically distributed (id) from the same probabilistic law (e.g., Gaussian, Gumbel, Weibull etc). As probabilistic laws are continuous, the stationarity analysis also addresses the problem of continuous execution times, even though execution times are discrete variables (see footnote 4). The stationarity analysis in the DIAGXTRM applies a test to check Definition 5.1.

As in practice the law of the execution times is unknown, we consider that a trace of execution times, at each discrete time instant t, can be written as the sum of a deterministic trend $f(t)$, a random walk r_t and a stationary residual ϵ_t [27]:

$$\mathcal{T}(t) = f(t) + r_t + \epsilon_t. \tag{4}$$

r_t is a random walk and may be written $r_t = r_{t-1} + u_t$ where u_t is a noise following a Gaussian distribution of mean 0 and unknown standard deviation σ_u. The Kwiatowski Philips Schmidt Shin (KPSS) test [27] checks whether $\mathcal{T}$ has a null deterministic trend and a null random walk for stating $\mathcal{T}$. In the case of a null deterministic trend, the KPSS test consists in testing the null hypothesis $H_0 : \sigma_u = 0$.

The KPSS test is applied to $\mathcal{T}$ and its confidence level is evaluated on the basis of the KPSS *result* and the associated $p - values$ as in Section 4.1, of the test detailed in [27].

5.2 Independence analysis

The independence analysis focuses on the short range dependence that refers to Berman's condition, or condition D in [29, 30]:

CONDITION 1 ($D(u_n)$). *For any integers p,q and n:* $1 \leq i_1 < \ldots < i_p < j_1 < \ldots < j_p \leq n$ *such that* $j_1 - j_p \geq l$ *we have*

$$|P(\{C_i, i \in A_1 \cup A_2\} \leq u_n) - P(\{C_i, i \in A_1\} \leq u_n) P(\{C_i, i \in A_2\} \leq u_n)| \leq \alpha_{n,l}, \tag{5}$$

where $A_1 = \{i_1, \ldots, i_p\}$, $A_2 = \{j_1, \ldots, j_p\}$ *and* $\alpha_{n,l} \to 0$ *as* $n \to \infty$ *for some sequence* $l = l_n = o(n)$.

For *distant enough* measurements, here l as the distance, and with u_n a sequence in the Fisher-Tippett-Gnedenko theorem, Condition 1 assures that the limit law of the maxima is still a GEV. In this view, blocks of execution time measurements of length l are considered, and their degree of correlation is evaluated with the Brock Dechert Scheinkman (BDS) test [11]. By choosing different values of length l, the degree of correlation varies and enables to identify particular patterns within the trace of execution times; Condition 1 holds for decorrelated blocks. The BDS test consists in testing the null hypothesis $H_0 : \mathcal{T}$ *is an iid trace* [11, 34] on the basis of the correlation integral. It allows to evaluate the statistical relationship between consecutive measurements (independence) and if they belong to the same distribution (identical distribution).

DEFINITION 5.2 (CORRELATION INTEGRAL). *The correlation integral* $CI_{l,n}(\epsilon)$ *at embedding dimension* l *for a distance* ϵ *is*

$$CI_{l,n}(\epsilon) = \frac{1}{\binom{n}{2}} \sum_{1 \leq s < t \leq n} \sum \chi_\epsilon(||C_s^l - C_t^l||). \tag{6}$$

For an iid trace $\mathcal{T}$:

$$\forall\ l,\ \epsilon,\ CI_l - CI_1^l \simeq 0\ for\ n \to \infty. \tag{7}$$

The correlation integral measures the degree of correlation between patterns (C_s^l and C_t^l) of different lengths l within $\mathcal{T}$ depending the absolute distance ϵ and if it tends to the correlation integral of 1-length patterns (CI_1) to the power of l then the short range dependence is accepted. The *result* of the BDS test follows a Gaussian law of mean 0 and standard deviation 1 giving the critical values [11] and so the associated confidence levels as in Section 4.1. The BDS test is applied in practice as in Algorithm 2.

Algorithm 2 Application of the BDS test [10]

```
1: procedure INDEPENDENCEANALYSIS($\mathcal{T}$)
2:     for $\epsilon \in \{\frac{1}{2}sd(\mathcal{T}), sd(\mathcal{T}), \frac{3}{2}sd(\mathcal{T})\}$ do  ▷ Correlation distance
3:         for $l$ from 2 to $\frac{length(\mathcal{T})}{200}$ by 1 do  ▷ Embedding dimension
4:             $results.append$(DEFUZZIFICATION ($BDS(\mathcal{T}, \epsilon, l)$))  ▷ $results$ is a list of the BDS test results
5:         end for
6:         $cl_2 \leftarrow \frac{sum(results)}{length(results)}$  ▷ Aggregation of the results
7:     end for
8: end procedure
```

5.3 Extreme independence

The reliability of the statistical model of the extreme execution time measurements depends on their independence. The extreme independence analysis depends on the selected threshold u like in Figure 3 that gives the peaks of execution time and stresses the presence of unreliable peaks that directly impact the GPD law. For instance, if an extreme burst of measurements occur, like many tasks running in parallel on different cores trying to access a memory unit at the same time, then all the measurements in or close to the burst depends on this same rare event. The peaks close to the burst are dependent endangering the pWCET estimates reliability, as formalized in condition D' [29, 30]:

CONDITION 2 $(D'(u_n))$. *The relation*

$$\lim_{i\to\infty}\limsup_{n\to\infty} n \sum_{j=1}^{[n/i]} P(C_1 > u_n, C_j > u_n) = 0, \quad (8)$$

has to be verified.

Condition 2 imposes that for one measurement over the threshold then the probability the following execution times are over the same threshold has to tend to zero. If the relation holds for the trace of execution times then the peaks over the threshold are independent i.e. there is no cluster of extreme execution times. It is important to note that the relation highly depends on the selected threshold.

The extreme index (EI) θ, $\theta \in]0;1]$ [20], indicates the degree of clustering of the peaks over the threshold. The EI expresses the probability to have a peak distant enough from another peak to be independent. In presence of bursts of peaks this probability decreases in function of the bursts size. The more the peaks are distant from each other the more the probability and so the more EI is near 1. According to the GPD idea, the proability of occurence of a peak decreases exponentially leading to an estimator of θ [21].

We build the set of inter-arrival times T_i, defined by the amount of measurements between two peaks, that follow an exponential process of intensity θ. The distribution of the inter-arrival times provides the unbiased estimator [21]:

$$\theta = \frac{2\left(\sum_{i=1}^{k-1}(T_i - 1)\right)^2}{(k-1)\sum_{i=1}^{k-1}(T_i-1)(T_i-2)}, \quad (9)$$

with k the number of peaks over the threshold also called the tail sample fraction standing for the number of execution time measurements that belong to the tail distribution.

Condition D' is a fuzzy condition, such that θ has to be near 1 in order to accept it. Hence, there is no systematic condition to conclude about the value of θ so that we define the confidence levels in Table 1 as in Section 4.1.

$\theta \in$	[1.00;0.95[	[0.95;0.90[	[0.90;0.85[	[0.85;0.80[	[0.80;0.00[
cl_3	4	3	2	1	0

Table 1: Confidence Levels of the Extreme Index.

As θ is the inverse of the mean size of the clusters, choosing 0.80 as the least bound on θ prevents from big clusters, so from unreliable pWCET estimates.

5.4 GPD parameter estimate and model matching

Independent peaks are extracted from the trace of execution times relatively to the selected threshold u, and are stored in a list *peaks*. It rests to estimate the parameters ξ and α_u of $F_{\mathcal{C}^\lambda}$ in Equation (3). For this purpose, we choose the *Maximum Likelihood Estimation* [20] method that performs well as it converges to convenient parameters.

Considering the set $\lambda = \{\xi, \alpha_u\}$ of the parameters to estimate according to a GPD, the *Maximum Likelihood Estimation* is an optimization problem that consists in exploring values of ξ and α_u and find λ that maximizes:

$$\ell(\lambda, exc) = \begin{cases} \ln \prod_{i=1}^{k} \frac{1}{\alpha_u}\left(\xi \times \frac{peaks[i]-u}{\alpha_u} + 1\right)^{-\frac{\xi+1}{\xi}} & if\ \xi \neq 0, \\ \ln \prod_{i=1}^{k} \frac{1}{\alpha_u} \exp\left(-\frac{(peaks[i]-u)}{\alpha_u}\right) & if\ \xi = 0. \end{cases} \quad (10)$$

Once the extreme execution time measurements have been fitted with a GPD it is necessary to check whether they really come from a GPD. To do so we introduce a matching test based on a quadratic statistic because it measures the square distance between the empirical CDF of the extreme measurements and $F_{\mathcal{C}^\lambda}$ estimated previously. The *Cramer Von Mises criterion* (CVM) performs well in the case of Extreme Value Distributions [28, 39] because it detects well whether the measurements come from the chosen distribution. The *result* of the CVM test measures the *distance* between the empirical distribution of the measurements and the pWCET estimate according to a GPD; a *distance* is defined as the *result* of the CVM test. Thus, the nearer zero the *distance* it is the better the GPD fits the extreme measurements, hence the more reliable the model it is. For applying the CVM test, the extreme measurements are sorted increasingly in a list *uom* for upper-ordered measurements such that:

$$distance = \sum_{i=1}^{k}\left(F_{\mathcal{C}^\lambda}(uom[i]) - \frac{2i-1}{2\times k}\right)^2 + \frac{1}{12\times k}. \quad (11)$$

Critical values of the CVM test are detailed in [16].

From the reliable pWCET estimate $\mathcal{C}^\lambda$ it is possible to derive WCET thresholds for a desired risk probability p. Formally, WCET thresholds are defined as the tuple $\langle WCET; p\rangle$ such that $p = P\left(\mathcal{C}^\lambda > WCET\right)$. For a desired risk probability e.g., 10^{-9} in aeronautics certification, the WCET threshold is directly given by [20]:

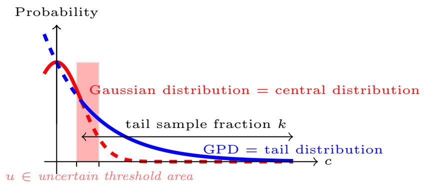
$$WCET = \begin{cases} u + \frac{\alpha_u}{\xi}\left(\frac{n}{k}p^{-\xi} - 1\right) & if\ \xi > 0, \\ u - \alpha_u \log(\frac{n}{k}p) & if\ \xi = 0, \\ u - \frac{\alpha_u}{\xi} & if\ \xi < 0. \end{cases} \quad (12)$$

The rationale of the WCET thresholds lies on two pilars: measurements on the real-time system, and the applicability of the EVT in order to infer the probabilistic law of extreme execution times. For very low risk probabilities e.g., 10^{-9}, WCET thresholds may not appear in reality, they only exist on the basis of the mathematical rationale of the EVT, which is more rationale than adding a percentage to the maximal execution time measurement. In the case $\xi < 0$, the risk probability zero exists, so that the WCET is deduced. In static analyses the difficulty is to have a complete model of the system; wrong or non complete models endangers the estimate confidence, while the proposed approach directly faces the real system. Furthermore, the probability of the WCET threshold also depends on the probability of the execution conditions e.g., input task parameters, that lead to the execution time measurements.

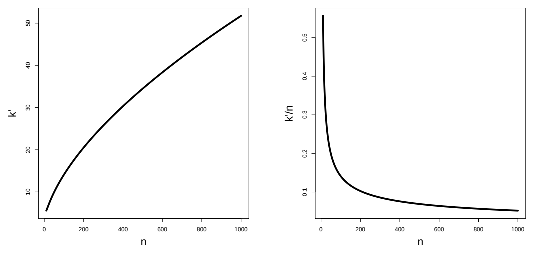
5.5 Threshold selection

The peaks of execution time highly affect the pWCET estimate because the estimate has to fit the peaks according to a GPD. The threshold is a great source of uncertainties as for different values correspond to different pWCET estimates increasing the uncertainty around the best estimate. Reviews for the threshold selection refer to many approaches [38] and still there is no systematic process for the selection. The threshold u is then a critical parameter because it directly provides the tail sample fraction k used for the parameter estimate, and impacting the reliability of the pWCET estimate.

We focus on the tail sample fraction to select the peaks such that the pWCET estimates uncertainty is minimized. To ensure tail convergence, k has to verify the two conditions, $\lim_{n\infty} k = \infty$ and $\lim_{n\infty} k/n = 0$ [38]. Hence, the tail sample fraction has to be small relatively to all the measurements in the trace and big enough to ensure the convergence of the limit law of the maxima. Moreover, for small values of k the GPD parameters vary a lot in function of k, whereas for greater values of k the parameters are biased by the amount of measurements, this is the bias-variance problem [20]. The threshold selection problem is resumed in Figure 6, where the central law of the measurements is a Gaussian distribution while the tail distribution is a GPD. The problem is then to estimate the right amount of execution time peaks by selecting the right threshold which lies in the uncertain threshold area. Thus we can only consider the tail distribution and not the central one. The existence of the right threshold relies on the hypothesis that the tail distribution of execution times converges well to a GPD.


Probability
Gaussian distribution = central distribution
tail sample fraction k
GPD = tail distribution
c
$u \in$ uncertain threshold area
Figure 6: The threshold selection problem, [14].

One specification of the DIAGXTRM tool is to be fully automatic so that we choose to apply a computational method, based on the respect of the CVM criterion. To converge to the right amount of execution time peaks, we first scope a potential area based on a rule of thumb $k' = \frac{n^{2/3}}{\log(\log(n))}$ [32] ensuring above conditions of convergence as showed in Figures 7(a) and 7(b).



(a) k' in function of n. (b) k'/n in function of n.

Figure 7: Plots of the rule of thumb k' in function of the length n of the trace of execution time measurements.

An uncertain area is drawn around k', where the right threshold should lie. The number of execution time peaks varies in the interval $k \in [k_{low} = \lfloor 0.5 \times k' \rfloor ; k_{up} = \lfloor 1.5 \times k' \rfloor]$ to explore other thresholds and still to cope with conditions of convergence. The threshold u is a function of the tail sample fraction k given by the quantile function q. q is a function of a percentage (between 0 and 1) and returns the execution time such that the desired percentage of measurements is below the returned execution time. Hence, $u = q(1 - k/n)$. Then, the peaks over u are fitted with a GPD giving the pWCET estimate and the distance between the experimental peaks and the pWCET estimate is evaluated with the CVM test. Finally, we iterate on k to cover the whole uncertain area.

The matching *result* given by the CVM test is a good indicator for selecting the right threshold because it indicates whether the execution time peaks really come from a GPD. Consequently, if the matching test gives a high confidence level for a threshold then we this threshold should be selected. To cope with conditions of convergence, a preference is added for thresholds given by a tail sample fraction close to k'. The matching test is so reduced from 4 to 3 confidence levels and a bonus in $[0; 1]$ is added depending on the value of k. The bonus evolves linearly from 0 to 1 in $[k_{low}; k']$ and from 1 to 0 in $[k'; k_{up}]$. To sum up, the computation of the confidence level for solving the threshold selection problem is presented in Algorithm 3.

Algorithm 3 Confidence level algorithm for the threshold selection

```
1: procedure ThresholdSelection($k$, $distance$)
2:     $cl_4 \leftarrow$ Defuzzification ($distance$)        ▷ $\in \{0; 1; 2; 3\}$
3:     if $k \geq k'$ then
4:         $cl_4 \leftarrow cl_4 + \frac{1}{1-\frac{k'}{k_{up}}} - \frac{\frac{1}{1-\frac{k'}{k_{up}}}}{k_{up}} \times k$        ▷ $\in [0; 4]$
5:     else
6:         $cl_4 \leftarrow cl_4 + \frac{1}{1-\frac{k'}{k_{low}}} - \frac{\frac{1}{1-\frac{k'}{k_{low}}}}{k_{low}} \times k$        ▷ $\in [0; 4]$
7:     end if
8: end procedure
```

As main contributions of the paper focus on the logical workflow and the decision making process regarding the applicability of the EVT, evaluations of Algorithm 3 will be the subject further investigations.

6. APPLICATION

In this section, DIAGXTRM is applied to a case study where the considered task is the *fibcall* from the Mälardelen benchmark [3]. To it, we intend finding its pWCET estimates in different execution conditions. The defined case study represents an example that could be done in an industrial declination. The *fibcall* task computes the i^{th} term in the Fibonacci sequence by a *for* loop implementation, with $i \in [2; 30] \cap \mathbb{N}$ so that there is no infinite loop. The set of possible inputs is denoted by $IN = \{i,\ i \in [2; 30] \cap \mathbb{N}\}$. The whole DIAGXTRM tool is implemented in R.

Hardware Platform.

The platform running the *fibcall* task has two Intel®Xeon®E5620 2.4 GHz sockets, each one with four cores and three levels of cache. The first two levels (L1 and L2) are private to each core, while the last level (LLC, equivalently L3) is shared to the cores belonging to the same socket.

Execution Conditions.

The task is implemented in C and runs periodically on one core; no interrupt (Irq) are present on the running core as they are redirected to other cores with Python system programming. To guarantee the real-time task execution, we set its scheduling policy to the Linux SCHED_FIFO policy. The *fibcall* task is executed under different conditions to explore systemic effects (congestion and interference from shared resources) that can affect extreme execution times:

Scenario 1 $S1$**:** *fibcall* is executed in isolation, it represents the case with no task interference and the reference scenario to compare with.

Scenario 2 $S2$**:** *fibcall* is executed with the task *cnt* [3] on the same core, one after the other. *cnt* counts non negative numbers in a $10^4 \times 10^4$ matrix. Such a large data structure is applied to create interferences at different cache memory levels to *fibcall*.

Scenario 3 $S3$**:** *fibcall* is executed with the task *cnt* in parallel on a different core that shares a LLC with the core where *fibcall* runs. Thus no interference within the same core but interference through shared resources.

Scenario 4 $S4$**:** a combination of $S2$ and $S3$ with two *cnt* tasks. One *cnt* on the core where *fibcall* runs, and another *cnt* that runs in parallel on core sharing a LLC with the core where *fibcall* runs. Each *cnt* task explores its own matrix to create interferences at different cache memory levels and avoid concurrent problems.

The scenarios may correspond to different choices of tasks repartition in a safety-critical embedded system, and the objective is to cope with both aspects of timing performance and safety by respecting strictly given timing constraints. The experiment consists in executing 500 times the fibcall task according to each execution condition presented above. The longest experiment time is approximately 20 minutes due to the execution of *cnt* that has to be allowed to explore the whole $10^4 \times 10^4$ matrix in scenarios $S2$, $S3$ and $S4$. Task inputs in IN are imposed iid according to a Uniform law during the experiment as they are generated randomly by the random C function at each time instant.

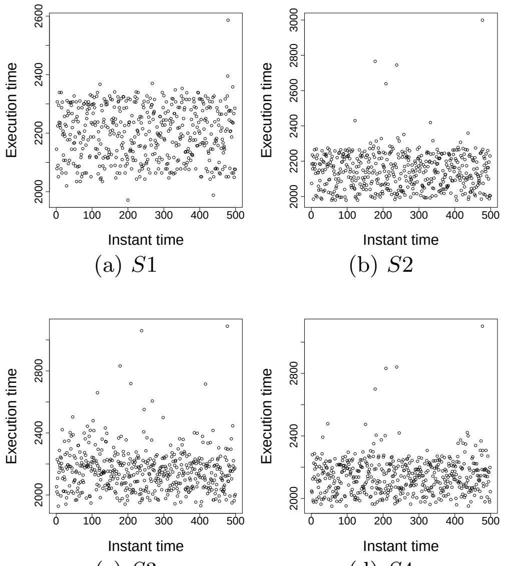
Results.

We now present the results of the experiments where execution times are measured in number of CPU cycles.

A first look at the traces in Figure 8 shows the repartition of the measurements and their randomness because there is no deterministic pattern over the time instants. Approximately, average execution times are between 2000 and 2300 CPU cycles. Measurements in the $S1$ case are concentrated in the average interval, while in the other cases, some measurements randomly deviate from the average interval.

ETPs in Figure 9 confirm the different repartitions observed in each trace, and a more important presence of extreme execution times in $S2$, $S3$ and $S4$ than in $S1$. Each execution differs from another one by only a few *for* loops, at most 28, explaining the concentration of measurements in an average interval. The interferences introduced with task *cnt* appear clearly in the ETPs as some measurements deviate from the average interval.

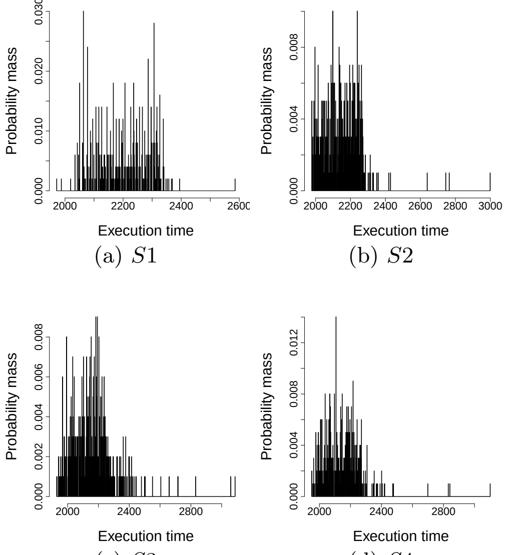
diagXtrm is applied to every trace of execution times for deriving the pWCET estimate and evaluating its reliability for each scenario. The tool gives the modeling results of the extreme execution times in Table 2 and also the EVT applicability results in Table 3 for the reliability of the estimates. The selected thresholds u are between 2500 and 2400 CPU cycles right at the frontier with the average interval highlighted with the ETPs and traces. Hence, only the extreme execution times, which are outside the average interval, are used for the GPD parameter estimate. Every shape parameter ξ is strictly greater than 0, and the minimal one is $S1$'s which



(a) $S1$ (b) $S2$

(c) $S3$ (d) $S4$

Figure 8: Trace of execution time measurements for every execution condition.



(a) $S1$ (b) $S2$

(c) $S3$ (d) $S4$

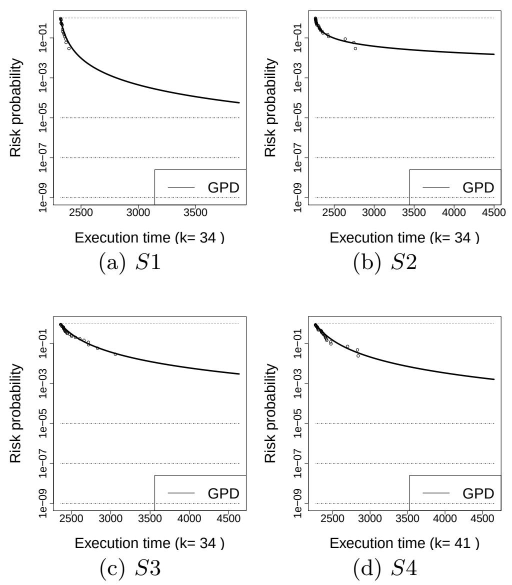
Figure 9: Experimental ETP for every execution condition.

is close to zero. By setting the risk probability p at 10^{-9}, as in aeronautics certification, the WCET threshold is deduced on the basis of parameters ξ and α_u for each respective scenario as in Equation (12). The greater ξ is, the more the WCET threshold diverges from the measurements; the greatest WCET threshold is $S2$'s which is around 10^7 times the respective maximal measurement. Estimated distributions of the extreme execution times are presented in Figure 10, showing the distribution convergence for each scenario.

All the traces are stationary (cl_1), with at least a high confidence and short-range independence (cl_2) is also verified for all the traces, as well as extreme independence (cl_3). Extreme

Trace $\mathcal{T}$	ξ	α_u	max	u	$\langle WCET; 10^{-9}\rangle$
S1	0.388	13.959	2586	2319.204	41719.696
S2	1.18	18.845	2999	2265.068	27960307288.975
S3	0.425	89.866	3088	2360.136	453591.23
S4	0.394	81.389	3102	2269.164	272528.444

Table 2: EVT results for every execution condition.



(a) $S1$ (b) $S2$ (c) $S3$ (d) $S4$

Figure 10: pWCET estimate for every execution condition.

independence is less obvious is S_4 but still accepted. The threshold selection criterion (cl_4) indicates also a high level of confidence for all the traces, because all matching confidence levels are strictly greater than 3. Algorithm 3 has the advantage to select the right threshold, if it exists, and to provide a confidence level about the distribution chosen for modeling the extreme execution times. Finally, the aggregation of all the confidence levels gives the reliability ($cl_{reliability}$) of the pWCET estimates, which are all strictly greater than 3, except for $S4$ that is quite near 3, then the pWCET estimates are highly reliable for all the scenarios.

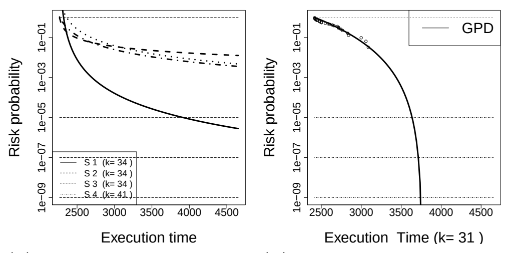
Trace $\mathcal{T}$	cl_1	cl_2	cl_3	cl_4	$cl_{reliability}$
S1	4	2.667	4	3.975	3.66
S2	4	4	3	3.975	3.744
S3	3	3.333	4	3.975	3.577
S4	4	2.333	1	3.604	2.734

Table 3: Confidence levels for every execution condition.

Originally, the distribution used for modeling the pWCET estimate is the Gumbel distribution ($\xi = 0$) by applying the block maxima approach. Within the DIAGXTRM this original approach may be evaluated by selecting block maxima instead of peaks and fitting a Gumbel GEV instead of a GPD. In this case study, given Theorem 3.1, the Gumbel distribution would be acceptable for the first scenario where ξ is near 0, and, as the Gumbel distribution converges to 0 faster than the Frechet ($\xi > 0$) distribution, it would decrease the pessimism of the WCET thresholds for the first scenario.

DIAGXTRM derives the pWCET estimate that best fits the peaks of execution time measurements for each scenario and all are diagnosed as reliable regarding the EVT applicability, however, the WCET thresholds of *fibcall* are all different. As pWCET estimates $\mathcal{C}^{\lambda,S2}$, $\mathcal{C}^{\lambda,S3}$ and $\mathcal{C}^{\lambda,S4}$ of scenarios $S2$, $S3$ and $S4$ converge slower than $\mathcal{C}^{\lambda,S1}$, they are more pessimistic due to the introduced interferences. Safety considerations would retain $\mathcal{C}^{\lambda,S4}$ as *fibcall* pWCET because $\mathcal{C}^{\lambda,S2} \succeq \mathcal{C}^{\lambda,S4} \succeq \mathcal{C}^{\lambda,S3} \succeq \mathcal{C}^{\lambda,S1} \succeq \mathcal{C}^{\lambda,S4}$ as shown in Figure 11(a), giving a WCET threshold of 2.796×10^{10} CPU cycles. However, the retained WCET threshold is more than 10^5 times greater than the WCET threshold in isolation which is 3500 CPU cycles questioning the rationale of this estimate. In scenarios $S2$, $S3$ and $S4$ the WCET threshold in isolation has more chances to be exceeded and respective probabilities to exceed it are 10^{-3}, 10^{-8} and 10^{-6}.

As interferences foster the appearance of extreme execution times, we gather the extreme execution times of all the scenarios. Let $\mathcal{T}^{\cap S}$ be the trace of extreme execution times of all the scenarios, then measurements are independent and stationary according to the diagnostic results and it is then possible to apply the EVT to $\mathcal{T}^{\cap S}$. Length of $\mathcal{T}^{\cap S}$ is 143 execution times so that the ideal tail sample fraction is 17 extreme execution times. The distribution of the extreme execution times of $\mathcal{T}^{\cap S}$ is deduced by applying the THRESHOLDSELECTION as in Algorithm 3 as shown in Figure 11(b). The final number of extreme execution times (31) is greater than the number given by Algorithm 3 (17), so that the distribution converges up to a risk probability of 10^{-9}. The matching confidence level (cl_4) of the model is equal to 4, so that the pWCET estimate $\mathcal{C}^{\lambda,\cap S}$ is fully accepted. As in this case $\xi < 0$, the WCET threshold for a null risk probability exists and is equal to $3764 (= \lceil 3763.446 \rceil)$ CPU cycles.



(a) Comparison of scenarios. (b) The tail of the tail distributions.

Figure 11: Plots including the four scenarios.

Under the hypothesis of non infinite blocking time of the task, the case $\xi < 0$ makes sense because the execution of *fibcall* has to end. By gathering larger amounts of extreme execution times from scenarios with different interference sources, estimates will refine the worst-case estimate. The degree of convergence of the pWCET estimate, given by ξ, indicates the impact of the introduced interferences, such that $S2$ is the scenario that impacts the most *fibcall* compared to the others. As a conclusion, *fibcall* WCET for the considered hardware platform is 3764 CPU cycles.

7. CONCLUSION

This paper presents the first sytematic and reproducible process for MBPTA approaches, with a logical workflow named DIAGXTRM, for applying the EVT to traces of execution times and deriving the pWCET of a task as well as its associated reliability. The systemic complexity of real time systems with non deterministic platforms (both time-randomized and non time-randomized) requires the use of MBPTA approaches to derive the pWCET of a task. The reliability of the pWCET estimates in MBPTA approaches depends on the theoretical hypotheses of the EVT that have to be tested. Results of statistical tests are often fuzzy and it becomes hard to make a decision on their basis requiring the introduction of a metric that indicates the fulfillment of a hypothesis. Execution conditions that provide the execution time measurements directly impacts the pWCET estimate so that MBPTA requires conditions that foster extreme execution times to refine the task pWCET.

8. REFERENCES

[1] http://www.proartis-project.eu/.

[2] http://proxima-project.eu/.

[3] *WCET project/ Benchmarks*, 2013.

[4] J. Abella, J. del Castillo, F. Cazorla, and M. Padilla. Extreme value theory in computer sciences: The case of embedded safety-critical systems. In *Proceedings 26th Euromicro Conference on Real-Time Systems (ECRTS14)*. IEEE, 2014.

[5] J. Abella, J. del Castillo, M. Padilla, and F. Cazorla. 68. extreme value theory in computer sciences: The case of embedded safety-critical systems. In *Current Topics on Risk Analysis: ICRA6 and RISK 2015 Conference*, page 579.

[6] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems*, 51(1):77–123, 2015.

[7] S. Altmeyer, B. Lisper, C. Maiza, J. Reineke, and C. Rochange. WCET and mixed-criticality: What does confidence in WCET estimations depend upon? In *15th International Workshop on Worst-Case Execution Time Analysis, WCET 2015, July 7, 2015, Lund, Sweden*, pages 65–74, 2015.

[8] K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar. WCET measurement-based and extreme value theory characterisation of CUDA kernels. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, page 279, 2014.

[9] G. Bernat, A. Colin, and S. Petters. pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. Technical report, 2003.

[10] A. Bonache and K. Moris. Chaos dans les ventes de biens à la mode et implication pour le contrôle de gestion. Post-print, HAL, 2011.

[11] W. A. Brock, J. A. Scheinkman, W. D. Dechert, and B. LeBaron. A Test for Independence based on the Correlation Dimension. *Econometric Reviews*, 15(3):197–235, 1996.

[12] J. J. Buckley. Fuzzy statistics: hypothesis testing. *Soft Comput.*, 9(7):512–518, 2005.

[13] A. Burns and B. Littlewood. Reasoning about the reliability of multi-version, diverse real-time systems. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, RTSS '10, 2010.

[14] J. Carreau and Y. Bengio. A hybrid pareto mixture for conditional asymmetric fat-tailed distributions. *IEEE Transactions on Neural Networks*, 20(7):1087–1101, 2009.

[15] F. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically analysable real-time systems. *ACM Transactions on Embedded Computing Systems*, 2011.

[16] S. Csorgo and J. J. Faraway. The exact and asymptotic distributions of cramér-von mises statistics. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 221–234, 1996.

[17] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *23nd Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2012.

[18] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. *Proceedings of the 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[19] S. Edgar and A. Burns. Statistical Analysis of WCET for Scheduling. In *RTSS*, pages 215–224. IEEE Computer Society, 2001.

[20] P. Embrechts, C. Klüppelberg, and T. Mikosch. *Modelling extremal events for insurance and finance.* Applications of mathematics. Springer, Berlin, Heidelberg, New York, 1997.

[21] C. Ferro and J. Segers. *Automatic Declustering of Extreme Values Via an Estimator for the Extremal Index.* EURANDOM, 2002.

[22] J. C. F. García and J. J. S. Méndez. A fuzzy logic approach to test statistical hypothesis on means. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence, 4th International Conference on Intelligent Computing, ICIC 2008, Shanghai, China, September 15-18, 2008, Proceedings*, pages 316–325, 2008.

[23] M. Gardner and J. Lui. Analyzing stochastic fixed-priority real-time systems. In *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 1999.

[24] E. Gumbel. *Statistics of Extremes.* Columbia University Press, 1958.

[25] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-Based WCET Estimation and Validation. In *9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*, pages 1–11, 2009.

[26] Y. Kato, M. Takahashi, R. Ohtsuki, and S. Yamaguchi. A proposal of fuzzy test for statistical hypothesis. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2929–2934 vol.4, 2000.

[27] D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root : How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1-3):159–178, 00 1992.

[28] F. Laio. Cramer-von Mises and Anderson-Darling goodness of fit tests for extreme value distributions with unknown parameters. *Water Resources Research*, 40, 2004.

[29] M. Leadbetter. On a basis for peaks over threshold modeling. *Statistics & Probability Letters*, 12(4):357–362, 1991.

[30] M. R. Leadbetter, G. Lindgren, and H. Rootzén. Conditions for the convergence in distribution of maxima of stationary normal processes. *Stochastic Processes and their Applications*, 8(2):131–139, 1978.

[31] M. Liu, M. Behnam, and T. Nolte. An evt-based worst-case response time analysis of complex real-time systems. In *8th IEEE International Symposium on Industrial Embedded Systems, SIES 2013, Porto, Portugal, June 19-21, 2013*, pages 249–258, 2013.

[32] M. Loretan and P. C. Phillips. Testing the covariance stationarity of heavy-tailed time series: An overview of the theory with applications to several financial datasets. *Journal of empirical finance*, 1(2):211–248, 1994.

[33] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A Trace-Based Statistical Worst-Case Execution Time Analysis of Component-Based Real-Time Embedded Systems. In *16th IEEE International Conference on Emerging Technology and Factory Automation (ETFA11), WiP session*, September 2011.

[34] R. Manuca and R. Savit. Stationarity and nonstationarity in time series analysis. *Phys. D*, 99(2-3):134–161, Dec. 1996.

[35] V.-A. Paun, B. Monsuez, and P. Baufreton. On the Determinism of Multi-core Processors. In C. Choppy and J. Sun, editors, *1st French Singaporean Workshop on Formal Methods and Applications (FSFMA 2013)*, volume 31 of *OpenAccess Series in Informatics (OASIcs)*, pages 32–46, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[36] M. Piera-Martinez. *Modélisation des comportements extrêmes en ingénierie.* Theses, Université Paris Sud - Paris XI, Sept. 2008.

[37] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *14th International Workshop on Worst-Case Execution Time Analysis*, pages 21–30, 2014.

[38] C. Scarrott and A. MacDonald. A review of extreme value threshold estimation and uncertainty quantification. *REVSTAT–Statistical Journal*, 10(1):33–60, 2012.

[39] M. A. Stephens. Goodness-Of-Fit for the Extreme Value Distribution. *Biometrika*, 64(3):583–8, 1977.

[40] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.

When the worst-case execution time estimation gains from the application semantics *

A. Bonenfant[2], F. Carrier[1], H. Cassé[2], P. Cuenot[4], D. Claraz[4], N. Halbwachs[1], H. Li[3], C. Maiza[1], M. De Michiel[2], V. Mussot[2], C. Parent-Vigouroux[1], I. Puaut[3], P. Raymond[1], E. Rohou[5], and P. Sotin[2]

[1]Univ. Grenoble Alpes, Verimag, France, first.last@imag.fr
[2]Univ. Toulouse, IRIT, first.last@irit.fr
[3]Université de Rennes 1/IRISA, first.last@irisa.fr
[4]Continental, first.last@continental-corporation.com
[5]Inria/IRISA, first.last@inria.fr

1 Introduction

Critical embedded systems are generally composed of repetitive tasks that must meet drastic timing constraints, such as termination deadlines. Providing an upper bound of the worst-case execution time (WCET) of such tasks at design time is necessary to guarantee the correctness of the system. Test based methods give realistic but unsafe results: they are never guaranteed to pinpoint the worst-case execution. On the contrary, static timing analysis methods compute safe WCET upper bounds, but at the cost of a potentially large over-approximation.

Over-approximation will lead to an over-calibration of the application resources, and even lead to defeat the scheduling of the tasks.

In static WCET analysis, a main source of over-approximation comes from the complexity of the modern hardware platforms: their timing behavior tends to become more unpredictable because of features like caches, pipeline, test prediction etc. Another source of over-approximation comes from the software itself: WCET analysis may consider as potential worst-cases executions that are actually infeasible, because of the semantics of the program and/or because they correspond to unrealistic inputs. For instance, in the automotive application (Engine Management System : EMS) of Continental Corporation the modules of the application are mostly implementing generic algorithms that used calibration data for possible adaptation. Moreover a theoretical worst case scenario could correspond to an unrealistic system state like high engine speed with low injection set point.

In the classical WCET estimation framework, the *data-flow analysis* is in charge of discovering infeasible execution paths. It must at least provide constant bounds for all the loops in the program, otherwise the WCET is not even guaranteed to be finite. Apart from loop-bounds, control-flow analysis usually identify simple semantics properties such as tests exclusions, that may prune infeasible execution paths when computing the WCET. These solutions remain largely ad-hoc, and there is no clear answer to the important questions raised by infeasible executions: What is the nature of such pruning properties? How to find them? (e.g., on the binary or the source code?) How to integrate them in a WCET estimation?

The goal of the W-SEPT project[1] is to define and prototype a complete semantic-aware WCET estimation workflow. It gathers researchers in the domain of timing and program analysis, together with an industrial partner from the real-time domain. The project mainly focuses on the semantic aspects, and thus, the pruning of infeasible paths. As far as possible, the idea is to extend and adapt the classical WCET estimation workflow, in particular, all that concerns the hardware analysis is inherited from previous work, namely the tool OTAWA[2].

Figure 1 depicts the proposed workflow. It retains the general organization of classical existing tools [16]. The bottom block is the WCET computation tool itself, organized in three steps: Control-Flow graph (CFG) construction, micro-architecture analysis, and worst-path search on the CFG. Gen-

*This work was funded by the Agence Nationale de la Recherche, project W-SEPT ANR-12-INSE-0001

[1]wsept.inria.fr

[2]www.otawa.fr

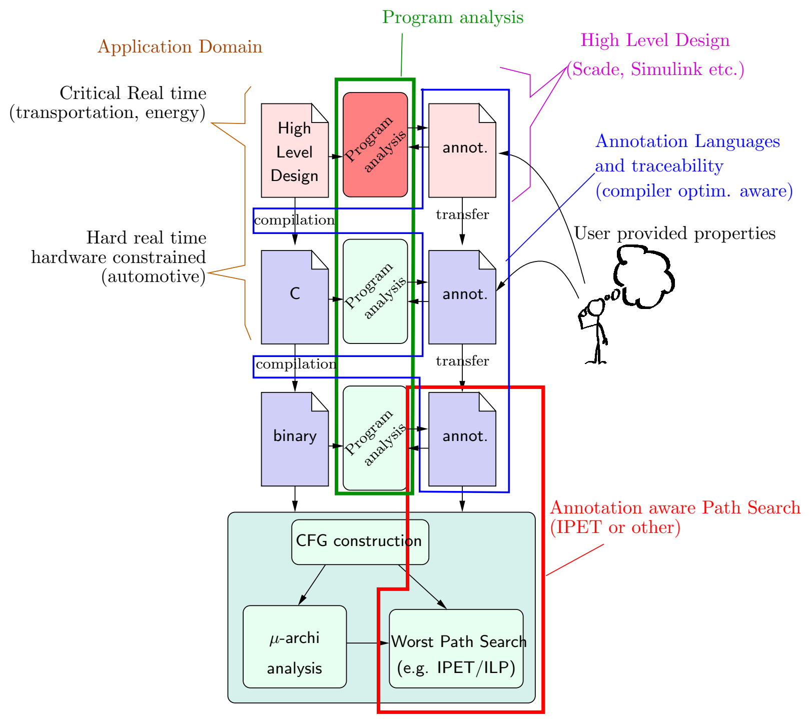


Figure 1: Work-flow and general organization of a semantic aware WCET estimation tool

erally, this last step uses the classical Implicit Path Enumeration Technique (IPET)[9]. This tool is fed by the binary code of the program, and a set of semantic informations classically named *annotation* file, and containing at least the loop bounds.

The (binary) annotations come from the data-flow analysis (we use here the more general term of *program analysis*). This analysis is generally performed at the source level (C language most of the time) rather than the binary level. Indeed, analyzing C code is technically much simpler than analyzing binary code, but more importantly, the analysis often requires extra information that only the human user can provide (e.g., inputs ranges, exclusion, implications). The user can probably express these properties in terms of the C variables, but it would be much harder or even impossible to do it in terms of the (compiled) binary code. This two-layers description raises the well known problem of *traceability* of annotations when transferring information between layers.

So far, the principles depicted in Figure 1 are rather classical. The project proposes first to take into account a third layer in the design flow: the use of high-level design languages tends to become common in the domains of (critical) real-time applications. Classical examples of high level design tools are Scade suite[3], used in avionics, energy or transportation, and Simulink/Stateflow[4] widely used in control engineering systems. These high-level design tools provide automatic code generation to C, which is no longer the source code, but only an intermediate code. A consequence is that user annotations and program analysis can be expressed and performed at the design level. Once defined this third layer, the project proposes to focus on three main issues depicted by enclosing boxes in the Figure 1 :

- Program analysis, that can be performed at design, C or binary level, and may take into account information provided by the user.
- Annotations and traceability between the language levels, strongly involve the compilers: as far as possible, the compilation process should be annotation-aware, in the sense that the program transformations performed by the compiler should be reflected as annotation transformations.
- (Worst) Path Search, must be adapted to take into account the (richer) kind of annotations produced by the workflow.

In this summary, we briefly introduce each step

[3]www.esterel-technologies.com/products/scade-suite
[4]mathworks.com/products/simulink/

of our workflow.

In section 2, we present how, at any stage, we can take into account annotations (from expert or automatically extracted) in order to produce a set of new ones. Then we automatically translate them when changing level, for instance loop unrolling, while keeping their validity regarding the code transformation/compilation.

In section 3, we describe how we adapted an WCET estimation tool in order to simplify, guide and even iterate the expert annotation process and exploit new kind of annotations.

One of the industrial goal is to prevent as early as possible in the development process the timing issues. In section 4 we detail the development cycle of an automotive application, and how some of the proposed solutions were experimented on a case study.

2 Find and trace useful information

In this section, we explain what kind of semantic properties may help to enhance the WCET estimation: where do they come from, which step of the application development do they refer to (binary, code, design), how are they transferred from one level to the next one. We consider two sources: annotations/feedback from expert and automatically extracted properties.

In order to express most of the properties, we use (and extend) FFX, an annotation language [20]. It is an open, portable and expandable annotation format. It allows combining flow fact information from different high-level tools. It is used as an intermediate format for WCET analysis.

2.1 Hypothesis and/or information from Expert

Some properties are known by the expert when considering the context of execution of the program: parameter domains, values for specific executions, parameters dependency... In classical tools [5] [6] [10] the expert input permits to reduce loop bounds. We use these precisions, called scenarios, in order to eliminate infeasible paths, in the execution context described by the expert.

Scenarios are used to give precisions on use cases: manual/automatic modes, context conditions like temperature, speed, height... Precisions that only an expert can provide because related to the context of execution of the program/application.

For these particular cases, when the expert wants to obtain a WCET estimation, it is possible to reduce the overestimation by taking into account constraints and conditions of execution. In most cases, information on these constraints allow to eliminate infeasible paths or bound more accurately the number of execution of certain part of the program. Indeed, when expert provides domain of certain parameters, our tools integrate these inputs and tighten our analysis.

The language FFX has been extended to express properties given by the expert. Limitations are due to the difficulties to make the expert write constraints in FFX. In order to resolve this issue, the expert expresses constraints in C and more recently the plug-in delta, describe in Sec 3.2, provides an interface. In a further work, we will define a format allowing the expert to address constraints directly in the code via comments.

2.2 Propagation and/or extraction of properties

2.2.1 Low-level

Looking for infeasible paths at binary level allows to benefit from the exact matching of the program with the hardware and to inject found properties immediately in the WCET computation. The price is an increase of analysis time caused by the program size and the loss of expressivity implied by machine instructions. Consequently, existing analyses either look for very simple infeasible paths [5, 15], or design a new WCET computation method [15]. Our approach tries to get rid of these limitations by using SMT solvers (Satisfiability Modulo Theories) to generate infeasible path properties.

2.2.2 Code level

The discovery of bounds and relations on numerical variables is a classical goal in program analysis [3, 4], the results of which can obviously be used to restrict the set of feasible paths considered in WCET evaluation. This can be helped by adding some counters to the code of the program: of course, adding a loop counter may result in finding a bound to this counter, and thus to the iteration number. Moreover, adding block counters, and finding relations between these counters can reveal subttle restrictions in the possible executions of the program. We illustrate this approach on a small example.

[5] www.absint.com
[6] www.bound-t.com

Program	LOC	#Cntr	#Inv	WCET init	WCET fin.	Improv.
selector	134	14	14	1112	528	52.6%
roll-control	234	25	19	501	501	0%
cruise-control	234	35	31	881	852	3.3%
even	82	9	8	2807	2210	23.3%
rate-limiter	35	2	2	43	29	32.6%
break	114	4	5	820	820	0%

Table 1: Improvement of OTAWA results with counter-based analysis at code level

Consider the following program fragment where x is not modified in block B1:

```
x = 0;
while c1 {
    if(x < 10){ B1: ...
    }
    if(c2){B2: x++; ...
    }
}
```

Let's add counters at important program points, e.g., counting the number of iterations in the loop (α) and the numbers of executions of blocks B1 (β) and B2 (γ):

```
x = 0; $\alpha = \beta = \gamma = 0$;
while c1 { $\alpha$++;
    if(x < 10){B1: $\beta$++;...
    }
    if(c2){ B2: $\gamma$++; x++;...
    }
}
```

An analysis of this instrumented program using an analyser of linear relations (here, we used the tool PAGAI [7]), automatically discovers that the following relations are always satisfied at the end of the program:

$$\gamma = x \ , \ \beta + \gamma \leq \alpha + 10 \ , \ \gamma \leq \alpha \ , \ \beta \leq \alpha$$

The inequality $\beta + \gamma \leq \alpha + 10$ is especially interesting, since it means that there are at most 10 iterations of the loop which execute both blocks B1 and B2.

Experiments: This approach has been implemented in a prototype tool [1], and applied in combination with OTAWA to several examples. Table 1 compares the results to those returned by OTAWA alone, for a set of small or medium-size programs. For each program it gives the number of lines of code, the number of introduced counters, the number of useful properties found by Pagai, the WCET evaluated by OTAWA alone, the WCET evaluated by OTAWA taking into account the properties, and the percentage of improvement.

2.2.3 High-level

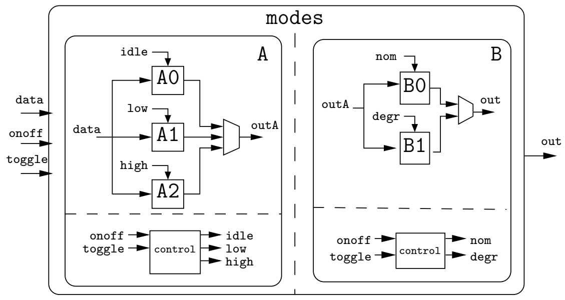


Figure 2: A typical high-level dataflow design

Critical embedded systems are often designed using an high level modeling language, such as Scade or Simulink. The system is then automatically compiled into classical imperative code (C in general), and then into binary code (cf. Fig 1).

Figure 2 shows a typical high-level data-flow design. For the sake of simplicity, it is represented as a diagram, while the actual program is actually written in Lustre [6], the academic textual language which is the ancestor of the industrial Scade language. This application consists of two sub modules, `A` and `B`, each of them consisting in two parts: a control part and a data processing part. The data processing part has different computation modes (e.g. `A0`, `A1` and `A2`), controlled by a *clock* (e.g. `idle`, `low` and `high`). An important property of such a design is that these modes are exclusive: at each reaction exactly one of the modes is activated. This information, obvious at the design level, may or may not be obvious at the C or binary level: depending on the compilation process, the (high level) mode exclusion may result or not into structurally exclusive pieces of code. In a more subtle way, we also know, for this particular program, that it exists a logical exclusion between the modes of the two sub-modules: if `A` is not `idle` (`A1` or `A2`), then `B` is necessarily in degraded mode (`B1`). This property is neither structural nor obvious: it is an *invariant* of the infinite cyclic behavior of the application, and, as a consequence, it is almost impossible to discover it at the low-level.

Based on these remarks, we have developed a prototype for discovering such properties, propagate them through the compilation process, and exploit them to enhance the WCET estimation. This prototype uses:

- an existing model-checker (Lesar [13]) to check the validity of properties at the Lustre level,
- a traceability module that can relate high level control variables (`idle`, `degr` etc.) to control points in the C code, and then control points in the binary; this traceability is partial (but safe): depending on compiler optimizations, some relations between high and low level maybe lost. However we had good results on this particular program, even with the higher level of optimization (option -O2 of the gcc compiler)
- the OTAWA tool-chain for he binary analysis and the construction of IPET (Implicit Path Enumeration Technique) problems, together with lp-solve to solve the IPET problems.

We have tried two strategies for enhancing the WCET.

Iterative algorithm:

- OTAWA is called for building an initial IPET problem, and lp-solve is called to find a first WCET control path candidate;
- according to the traceability information, the validity of this path is translated (if possible) into a logical condition on the high level variables (e.g. $\neg$`idle` $\wedge$ `low` $\wedge$ `nom`);
- Lesar is called to check this condition; if the condition is unsatisfiable, the WCET path candidate is proven unfeasible, the corresponding constraint is added to the IPET problem, and lp-solve is called again to find a new candidate, and so on. If the condition is satisfiable, the process stops with the current WCET.

Pairwise algorithm:

- The high level code is analyzed to find a set of interesting control variables, according to a simple heuristic: any Boolean variable that control computation modes (often called the *logical clocks*) are likely to control big pieces of binary code, and thus, have a big influence on the computation time. In the example, the five control variables are selected.
- We “blindly” search for all possible pairwise relations (either exclusions or implications) between these variables. For n variables, there are $4(n * (n-1)/2) = 2n(n-1)$ such (potential) relations (40 in the example). For each relations proven by Lesar, we generate the corresponding constraints at the binary level thanks to the traceability information; in the example, 5 over 40 relations are proven.
- OTAWA is called once with these constraints, and generate directly an enhanced WCET estimation.

optim.	reference		iterative		pairwise	
	wcet	cost	wcet	cost	wcet	cost
-O0	4718	64s	2371	163s	2372	67s
-O2	758	1s	457	5s	457	2s

Table 2: Exploiting high level properties: WCET improvement and computation cost (cpu second on a i7 workstation).

The whole experiment is presented in details in [14]; quantitative results are summarized and commented in Table 2 where two optimization levels and two strategies are experimented; enhancement is important for both level (-50% and -40%), and similar for both strategies. Iterative algorithm may be relatively costly, pairwise strategy has a constant overhead.

2.3 Traceability

Knowledge of semantic properties helps tighten WCET estimates. Such information is usually known at the design or source code level, whereas WCET estimation must be computed at the binary code level.

From design level to source code, we transfer the properties by tracing them in the code generator (by inserting additional comments in the C code).

From C to binary, hundreds of compiler optimizations may have a strong impact on the structure of the code, making it impossible to match source-level and binary-level control flow graphs. This ends up in a loss of useful information. For this reason, the current practice is to turn off compiler optimizations, resulting in low average-case and worst-case performance. To safely benefit from optimizations, we propose a framework to trace and maintain flow information up-to-date from source code to machine code [8].

The transformation framework, for each compiler optimization, defines a set of formulas, that rewrite available semantic properties into new properties depending on the semantics of the concerned optimization. Supported semantic properties are *loop bounds* and linear inequations constraining the execution counts of basic blocks. Consider, for example, loop unrolling, that replicates a loop body k times to reduce loop branching overhead and increase instruction level parallelism. The associated rewriting rule divides the initial loop bound by k, and introduces constraints on the execution counts

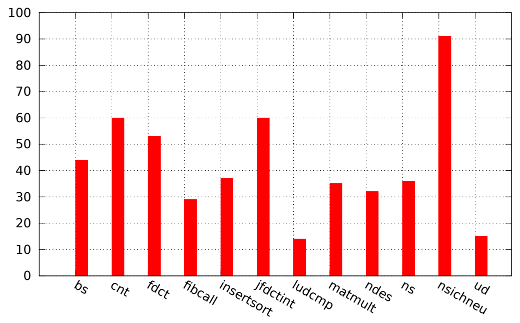


Figure 3: Impact of optimizations (`-O1`) on WCET. The y-axis represents the WCET with optimizations, normalized with respect to the WCET without optimization (`-O0`)

of the basic blocks within the loop (see [8] for details).

We implemented this traceability in the LLVM compiler infrastructure. Each LLVM optimization was modified to implement the rewriting rules corresponding to the optimization. Semantic information is initially read from a file in the FFX format [18] and then represented internally in the LLVM compiler and transformed jointly with the code transformations. Optimizations that do not modify the control flow graph can safely preserve the semantic information. Others must update the information to reflect the new graph. Note that, if a transformation happens to be too complex to trace the information, it can be disabled. This is a much better situation than the current practice which is disabling all optimizations.

Figure 3 reports the reduction of WCET estimates for codes from the Mälardalen benchmark suite [7], resulting from optimizations of level `O1`. In this experiment, only loop bounds are traced.

The experiments first show that it is technically feasible to transform all semantic information from C code to binary without loss of information. This is shown by the fact that we can compute the WCET of all benchmarks (a single missing loop bound would make the computation impossible). Secondly, we observe that option `-O1` yields an important reduction of estimated WCETs: 60 % in average, and up to 86 % (optimized WCET is 14 % of unoptimized) for benchmark *ludcmp*, which contains deeply-nested loops.

[7]www.mrtc.mdh.se/projects/wcet

2.4 Heuristic for targetting the "interesting" properties

In order to lower the real WCET, some approaches compute the criticality of piece of codes [2] or generate a static profile using probabilities for decisions at branching points [17]. The delta tool [19] aims at identifying the conditional statements that are unbalanced in terms of execution time weight (obtain so far by a naive account of instructions). This highlights, to the expert or the program analyzers, the parts of code where a semantic analysis or expert annotation should focus to gain more accuracy on the WCET estimation.

The following experiment is detailed in [19].

In the context of the case study, the expert initially provided a scenario of 30 parameter initializations (over 85 identified parameters). 54 Δ-conditions have been identified. 20 of the 30 parameters initialized in the provided scenario appear in the list of the Δ-conditions, 18 of them exhibiting the highest 10 Δ-values (difference of weight between the two branches) the list. 19 of the 54 Δ-conditions have low Δ-values (218 and less than 11) and no correspondence to the parameters in the scenario. As we rely on the parameter names to appear as operands in the Δ-conditions, a parameter may be linked to several Δ-conditions and vice versa.

Table 3 shows the result of WCET analysis of the module: column 1 lists the provided scenario, column 2 lists the number of specified parameters in the scenario and column 3 to 6 list the WCET estimate and improvement compared to the global WCET for an ARM7 lpc2138 platform, without and with a 1KB direct mapped data cache.

WCET analysis of the module without scenario, (1) global, reports 2553 as WCET estimate. WCET analysis of the expert-provided scenario, specifying 30 parameters, (2) full scenario, yields an improvement of 5%. Rows, (3)-(6), list the estimate and gain when specifying only those parameters involved in the i highest valued Δ-conditions.

To validate that specifications for parameters not contained in the list of Δ-conditions have little impact on the estimate, we supply the 10 parameter initializations that do not appear in any Δ-conditions, row (7).

Summarizing, branching statement analysis identified 20 of 80 parameters as important due to their high Δ-values in the list and they coincide with specified values in the expert-provided scenario. 10 parameters specified in the expert-provided scenario do not appear in the Δ-condition list and have almost no impact on the WCET estimate, while specifying only parameters identified in the

10 highest Δ-conditions still improves the estimate.

The experiment shows that our branching statement analysis can help system-experts focus on the relevant parameters from the vast number of possible parameters.

scenario	# param.	no cache	
(1) global, no scenario	0	2553	gain
(2) full scenario	30	2426	5%
(3) 3 highest Δ	3	2553	0%
(4) 8 highest Δ	10	2479	3%
(5) 9 highest Δ	14	2463	3.5%
(6) 10 highest Δ	18	2448	4%
(7) none of Δ	10	2551	0.08%

Table 3: WCET computation depending on parameters provided in scenarios

3 Integration in WCET estimation tool

In this section we explain how the information extracted in previous section may be exploited to enhance the WCET estimation. We show how they are taken into account into the WCET tool and how the expert or user may interact and get feedback from the WCET.

Scenarios and properties are given in FFX. The tool OTAWA is used to integrate all annotated property in the WCET estimation.

3.1 Exploitation through automata

In previous works, infeasible paths properties are encoded into integer linear programming constraints and taken into account at the last WCET estimation step [5]. In the project, we propose a general, versatile and non-intrusive process for integration of the paths properties[11, 12]. This process assumes that the WCET tool internally handles CFGs and integer linear constraints, which is the case of every IPET-based WCET analysers. The internal representation of the program is extracted, improved according to the annotations and set back in the tool. The transformation relies on a novel automata formalism that can represent both the program CFG and the annotations. The transformation itself is an automata product; its result is an automaton that allows only paths both existing in the original CFG and being valid with respect to the annotations. The analysis performed on the enriched CFG delivered a WCET improvement up to 10% on the benchmarks of the WCET Tool Challenge.

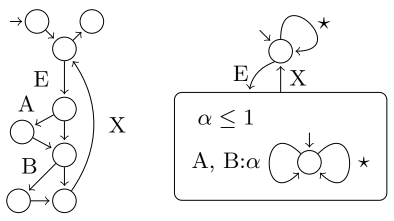


Figure 4: Path Property Automaton

The formalism, called Path Property Automata (PPA) offers the following features:

1. State based acceptance. Like in finite state automata, one can forbid some transitions according to the history of the execution.
2. Counter based acceptance. Before being accepted, a path must satisfy numerical constraints on the transitions it took.
3. Context of validity. The restrictions expressed using Features 1 and 2 can be subject to a context of validity. The notion of context is expressed in the formalism by hierarchical states.

Figure 4 contains two PPA. On the left, the PPA isomorphic to the program CFG. On the right, the PPA reflects the annotation "in each iteration of the loop starting with E and ending with X, at most one of A or B can be taken".

3.2 Iterative process from WCET tool to the user

Based on the delta tool, we have developed a graphical tool.

Figure 3.2 shows the iterative process: given a C program and a scenario, the delta tool provides annotations (in FFX format) and a list of δ-conditions. The Eclipse Delta Plugin, allows to easily visualize these δ-conditions and the parameters involved. The expert can re-define a scenario by visualizing the relevant parameters, obtain the consequent new unbalanced conditionals caused by the scenario, and iterate this process by refining properties on parameters in order to gain accuracy on the WCET estimate.

Figure 3.2 is an overview of the Eclipse Delta Plugin. In the center, the code is loaded. Lines corresponding to the selected δ-condition are highlighted. A list of related parameters is provided, allowing to refine the initial value. The adapted scenario is then automatically created as a FFX file. Either it is reloaded in order to identify other relevant branchings, either it is given to Otawa in order to compute the WCET estimate.

This plugin can also be used as an assistant to

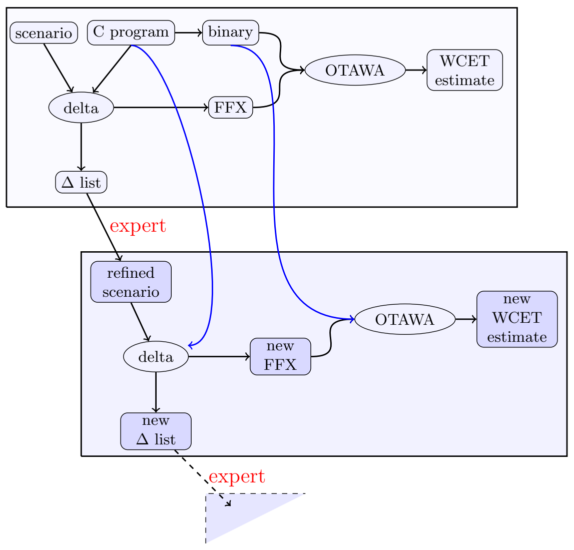


Figure 5: Scenario Refining Iterative Process

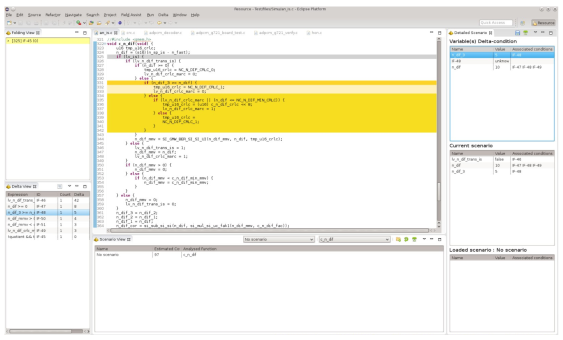


Figure 6: Delta Eclipse Plug In

create scenario.

4 An industrial case study

This section presents the application of some of the proposed techniques to a case study given by the industrial partner of the project, Continental. Only a part of the experiments are presented, because on one hand, the case study does not match some of the techniques — for instance, it is given in C, so the techniques devoted to higher-level code (§2.2.3) don't apply — and on the other hand, some further experiments are still to come.

The industrial case study for the experimentation of the new WCET method is an automotive application extracted for Continental industrial portfolio.

The Engine Management System (EMS) application is a complex real time application. The software application is an assembly of multiple sources:

- C Code generated from Simulink model using model based approach. A one to one relationship is established between Simulink subsystem and C source file.
- Manual C code for other functions.
- Third party software from customer. It can be a Simulink model, a source code or an object file.
- AUTOSAR execution platform (BSW) with either code developed internally or library code bought as third party software form the market.

The software is executed on dedicated microcontroller for embedded automotive (even Engine Systems) market. It requires the use of ”specific” target compiler (unlike usual GGC or LLVM complier), using an internal standardized configuration options (optimization scheme, in lining, cache control, memory allocation strategy ...).

The software module complexity and consequence on timing bound effects are managed by:

- applying encapsulation, modularity and portable design principle with focus on module reuse,
- defining generic module algorithm and use calibration data for possible adaptation. A calibration is a constant ROM which is configurable during development, and frozen for software production,
- applying MISRA-C coding rule that prevents use of dangerous coding (limitation of the use of pointer, implementing loop with bound,),
- abstracting hardware dependencies by a Hardware Abstraction Layer (hardware platform and compiler independence).

Moreover, the today software is designed and implemented to support multi-core architecture, but first we decided to ignore this constraint in the study.

The definition and sizing of the software architecture is driven by resource consumption limitation and safety related constraints. The co-engineering with customer requires defining common methodologies to be able to manage the resource such as: component split, memory control, timing control, OS and AUTOSAR services integration...

The timing resource is the one of most critical one. It needs to be estimated to organize a sound scaling of processor resource and for the task timing allocation. So, a generic schema for task scheduling is defined by an architecture team and feed by the project with all software module runnable units (executable part of a software module). This is the integration work. Such configuration shall be evaluated for the prediction of scheduling of the application and then verified by measurement on real HW target. Today schedulability design and evaluation are based on measurement data, stored in a database. At integration time, it is necessary to evaluate the runtime of runnable units integrated in the Tasks, in order to properly configure the Task / the integration. Usually, this evaluation is based on the measurements done at the end of the previous V-cycle. As real measurements on bench can only be done sporadically (e.g. once/month) compared to the continuous integration work (e.g. several steps / day), the measurement data from the database becomes rapidly obsolete, and needs to be replaced by estimation. In addition, software configurations and timing measurement conditions are very difficult to standardized and therefore to compare and reuse. The actual orientation for use of heuristics for prediction is then limited in term of granularity.

In addition, the strong reuse strategy is based on reusable software components out of the hardware development context. So, the timing performance of these components needs to be provided (and reused) with an abstract timing estimation (hardware dependence limited). Moreover, the WCET is important to determine, but not always represents a realistic execution due to software interactions complexity.

Continental in this project aims to find a solution for the early estimation of the time execution of software and to allow computing realistic WCET values. The sensitivity to the hardware core architecture must be established to validate the results of the estimation. Of course, this approach requires to be supported by a reliable methodology, capable to support customer/client engineering exchange.

A set of software components representative of the EMS were selected to evaluate the technologies represented in the workflow (Fig 1). The expert uses the annotation concept (section 2.1) to capture behavioral scenarios of the application. These scenarios match the operational conditions of execution of the software, which means real engine conditions. As an example, a theoretical worst case scenario could correspond to an unrealistic system state like high engine speed with low injection set point.

The expert is using heuristics (section 2.4) to describe the scenarios. Out of the general conditions, he concentrates his effort on main effect of large branches. In particular, it is not necessary to spend engineering work on determining an active branch, if the two alternatives have an equivalent weight. The runtime estimation is refined using the propagation of the previously defined properties, in addition to the resolution of the own SW code semantics with the help of eventual annotations. The property propagation at C level is mostly used for this estimation.

The property propagation at low level (HW, bin, asm) has been used as verification of the estimated runtime for one specific core architecture. The high level properties propagated from Lustre language (SCADE environment not used for EMS application) is seen as requirement for the Simulink C code generation chain. The tool environment (section 3.2) is used on the selected software module to estimate the timing execution of the runnable units of the software component.

For the selected component, the estimation of the software component timing execution is performed using the tool prototype environment (section 3.2). Finally, the traceability concept (section 2.3) couldn't be applied in our application due to specific target compiler used. It could lead to identification of new requirement for future embedded compiler.

5 Conclusion

In this paper, we introduce the workflow implemented in the W-SEPT project to better integrate the application semantics. We show that semantic properties may be found at each language level (design, source and binary), they have to be traced through the compilation steps to be taken into account in the WCET estimation. The current implementation already showed interesting results for benchmarks and real applications, and good feedback from our industrial partner.

References

[1] Remy Boutonnet and Mihail Asavoae. The WCET analysis using counters - a preliminary assessment. In *Proceedings of 8th JRWRTC, in conjunction with RTNS14*, Versailles, France, October 2014.

[2] Florian Brandner et al. Static profiling of the worst-case in real-time programs. In *RTNS*, pages 101–110, 2012.

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis

of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.

[4] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978.

[5] Jan Gustafsson et al. Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In *RTSS*, 2006.

[6] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[7] Julien Henry, David Monniaux, and Matthieu Moy. Pagai: A path sensitive static analyser. *Electr. Notes Theor. Comput. Sci.*, 289:15–25, 2012.

[8] Hanbing Li, Isabelle Puaut, and Erven Rohou. Traceability of flow information: Reconciling compiler optimizations and WCET estimation. In *22nd International Conference on Real-Time Networks and Systems, RTNS'14, Versailles, France, October 8-10, 2014*, 2014.

[9] Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(12), 1997.

[10] Björn Lisper. SWEET a tool for WCET flow analysis. In *ISOLA*, 2014.

[11] Vincent Mussot, Armelle Bonenfant, Pascal Sotin, Philippe Cuenot, and Denis Claraz. From relevant high-level properties to WCET computation improvement. In *ERTS2*, 2013.

[12] Vincent Mussot and Pascal Sotin. Improving WCET analysis precision through automata product. In *21st IEEE International Conference on Embedded Systems and Real-Time Computing Systems and Applications, RTCSA'15*, Hong Kong, August 2015.

[13] P. Raymond. Synchronous program verification with lustre/lesar. In S. Mertz and N. Navet, editors, *Modeling and Verification of Real-Time Systems*, chapter 6. ISTE/Wiley, 2008.

[14] Pascal Raymond, Claire Maiza, Catherine Parent-Vigouroux, Fabienne Carrier, and Mihail Asavoae. Timing analysis enhancement for synchronous program. *Real-Time Systems*, pages 1–29, 2015.

[15] Vivy Suhendra et al. Efficient detection and exploitation of infeasible paths for software timing analysis. In *DAC*, pages 358–363, 2006.

[16] Reinhard Wilhelm et al. The worst-case execution-time problem - overview of methods and survey of tools. *TECS*, 7(3), 2008.

[17] Youfeng Wu and James R. Larus. Static branch frequency and program profile analysis. In *MICRO*, pages 1–11, 1994.

[18] Jakob Zwirchmayr, Armelle Bonenfant, Marianne de Michiel, Hugues Cassé, Laura Kovács, and Jens Knoop. FFX: A portable WCET annotation language (regular paper). In *International Conference on Real-Time and Network Systems (RTNS), Pont-à-Mousson, 08/11/2012-09/11/2012*, pages 91–100, http://portal.acm.org/dl.cfm, novembre 2012. ACM DL.

[19] Jakob Zwirchmayr, Pascal Sotin, Armelle Bonenfant, Denis Claraz, and Philippe Cuenot. Identifying relevant parameters to improve WCET analysis. In *WCET*, 2014.

[20] Jakob Zwirchmayr et al. FFX: A portable WCET annotation language (regular paper). In *RTNS*, pages 91–100, 2012.