CAN Database Verification Framework Using UPPAAL

Beomyeon Cho, Taewook Kim, and Jin-Young Choi

***Abstract*—It is inevitable that the response time of a message is delayed in a CAN network where multiple ECUs share a single bus. However, delays of response time should be minimized for messages related to the safety of a driver or a vehicle. If the response time exceeds the deadline, the safety of the driver and the whole of the vehicle system may be impaired. Therefore, it is essential to verify the CAN database in advance so that the response time of a message does not exceed the deadline. In this paper, we propose a framework composed of UPPAAL and a DBC2XML component, which automatically generates a UPPAAL model from a CAN database and show how to verify the CAN database using this framework.**

***Index Terms*—Controller area network, DBC, formal verification, UPPAAL.**

I. Introduction

CAN (Controller Area Network) is a serial communication protocol that supports distributed real-time control and multiplexing, and is widely used as an automotive communication protocol globally [1]. There are several ECUs (Electronic Control Units) connected to one CAN bus, and each ECU periodically or aperiodically transmits various messages to the CAN bus. If multiple ECUs try to transmit a message at the same time, only one message with the highest priority is successfully transmitted through the arbitration process, and the others that fail to be transmitted are retransmitted. Due to this nature of CAN protocol, the response time of low priority messages is delayed. By setting an offset for each message, the CAN bus bandwidth can be utilized to the maximum, reducing the response time delay. However, using a large number of messages or allocating offsets inefficiently still results in a delay of response time.

In the case of messages containing information related to airbags or brakes, delay of response time can have a serious impact on the safety of the driver and the vehicle. That is, the delay of response time can damage the safety of the driver and the whole of the vehicle system. When designing a CAN network, these messages are given relatively higher priority than other messages, but there may be other messages in the same CAN network that should not be delayed. Therefore, the messages of the CAN network used in the vehicle, which is a safety-critical system, must be verified to ensure that the response time meets the given deadline.

There are studies on the delay of message response time [2]-[6]. Ken Tindell *et al*. use equations of the scheduling theory [2]. On the other hand, [3] and [4] are studies that verify various properties of a CAN network based on model checking using timed automata and temporal logic [7]-[9].

In [3], Jan Krakora *et al*. model the CAN network using UPPAAL, a formal specification and verification tool, and verify logical and timing properties for the model. However, because of the complicated model with low abstraction level, it takes a long time to verify one property, and verification is performed with only four messages in the paper. That is, there is a limit to verifying a real CAN database using many messages.

In [4], Can Pan *et al*. model the CAN network using UPPAAL and verify 11 given properties. In order to satisfy more properties, they propose a way to dynamically change the message IDs to raise the priority. However, in order to change the priority dynamically, the application layer's algorithm must be involved in MAC (Media Access Control), which is the role of the data link layer. Also, dynamically changing message IDs make other ECUs difficult to interpret the messages and require additional mechanisms to solve it.

In this paper, we propose a framework to automatically generate UPPAAL model from real CAN database which contains a large number of messages, and to verify logical properties and timing properties using the generated UPPAAL model. By using this framework, it is possible to verify the safety of the CAN database at the vehicle design stage, and to find counter examples which violate the properties in advance. We can design a safer vehicle by redesigning the CAN network to reflect these cases.

The paper is organized as follows: in Section II, we introduce the components that make up the framework. In Section III, by using the framework we demonstrate verifying a given DBC file, which is a CAN database and show the verification results. Section IV concludes the paper.

II. CAN Database Verification Framework

The proposed framework consists of DBC2XML component and UPPAAL as shown in the Fig. 1. DBC2XML accepts a CAN database file (.DBC) as input, and generates an UPPAAL model (.XML). The DBC file contains various information about the CAN network, and DBC2XML extracts only information about the message. DBC2XML generates an UPPAAL model based on the extracted information. UPPAAL

Manuscript received July 10, 2017; revised October 23, 2017. This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2015-0-00445) supervised by the IITP (Institute for Information & communications Technology Promotion.

Beomyeon Cho and Taewook Kim are with the Department of Automotive Convergence, Korea University, Seoul, Korea (e-mail: bycho@formal.korea.ac.kr, twkim@formal.korea.ac.kr).

Jin-Young Choi is with Graduate School of Information Security, Korea University, Seoul, Korea (e-mail: choi@formal.korea.ac.kr).

DOI: 10.7763/IJCTE.2017.V9.1182

performs model checking by inputting the UPPAAL model generated by DBC2XML and properties to be verified. The logical properties and timing properties used in this paper are included in the library, and can be added manually in UPPAAL if users want to further verify other properties. If the UPPAAL model satisfies the given property, it results in 'satisfied', and if not, it results in 'not satisfied' with a counter example.

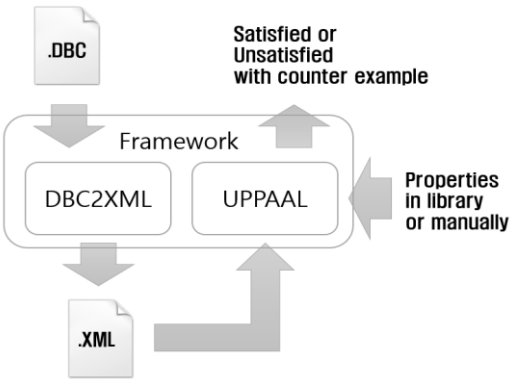

Fig. 1. Overview of proposed framework.

A. CAN Database

The CAN database is an ASCII file containing information about the CAN network and can include information about the ECUs, the messages, and signals used by each ECU. There are many kinds of CAN database, and the DBC format designed by Vector is the most widely used.

Among the information contained in the CAN database, information on a message generally includes a message name, an ID, a DLC (Data Length Code), and a cycle time, etc. If necessary, users can define and use message attributes additionally. The message information included in the DBC file to be used in this paper is shown in Fig. 2, and it is assumed that the deadline and the offset are not included in the DBC file.

Name	ID	DLC [Byte]	Cycle Time
ECU0_GW1	0x100	8	10
ECU1_POE1	0x101	8	10
ECU2_OE1	0x102	8	20
ECU3_OE2	0x103	8	20
ECU4_P1	0x104	8	20
ECU5_POE2	0x105	8	20
ECU6_POE3	0x106	8	10
ECU7_P4	0x107	8	0
ECU8_P5	0x108	8	0
ECU9_PIF1	0x109	8	10
ECU10_IF1	0x10A	8	5
ECU11_P2	0x10B	6	5
ECU12_P3	0x10C	8	5
ECU13_FP1	0x10D	8	20
ECU14_FP2	0x10E	8	10

Fig. 2. Example of DBC file.

B. DBC2XML

DBC2XML is a component that automatically generates an UPPAAL model (.XML) from a CAN database file (.DBC) to verify logical properties and timing properties. DBC2XML is executed in the following three steps, and they are repeatedly executed for all DBCs in the specified folder:

1) In the parsing step, DBC2XML extracts information about messages such as message name, ID, DLC, and cycle time from the DBC file and stores it in memory in the form of map data structure.
2) In the processing step, DBC2XML assigns deadlines and offsets, which are not defined in the DBC file but are necessary for verifying the properties, to each message and stores them in the map data structure. In this paper, we assume that the deadline of all messages is 1 ms, and the offsets are arbitrarily assigned in units of 1 ms. The message information after the processing step is shown in Fig. 3.

Name	ID	DLC...	Cycle Time	Deadline	Offset
ECU0_GW1	0x100	8	10	1	4
ECU1_POE1	0x101	8	10	1	4
ECU2_OE1	0x102	8	20	1	2
ECU3_OE2	0x103	8	20	1	3
ECU4_P1	0x104	8	20	1	4
ECU5_POE2	0x105	8	20	1	0
ECU6_POE3	0x106	8	10	1	4
ECU7_P4	0x107	8	0	1	2
ECU8_P5	0x108	8	0	1	3
ECU9_PIF1	0x109	8	10	1	4
ECU10_IF1	0x10A	8	5	1	0
ECU11_P2	0x10B	6	5	1	1
ECU12_P3	0x10C	8	5	1	2
ECU13_FP1	0x10D	8	20	1	3
ECU14_FP2	0x10E	8	10	1	4

Fig. 3. Message information after processing step.

3) In the making XML step, DBC2XML generates a UPPAAL model using the information contained in the map data structure and libraries located in the specified folder. The libraries include common contents for generating UPPAAL model independent of a given DBC file. Fig. 4 is a part of the DBC2XML source code that copies a library into an XML file and writes message ID information to the XML file. Fig. 5 is a part of the libraries necessary to generate Transceiver model.

```
// Copy global_decl library into .XML
fopen_s(&finput, "lib\\2_global_decl", "r");

while ((ch = fgetc(finput)) != EOF)
    fputc(ch, foutput);

fclose(finput);

// Write message IDs into .XML
fprintf(foutput, "const int id[%d] = {", N);

for (iter = msg_map.begin(); iter != msg_map.end(); ++iter)
    fprintf(foutput, "%d, ", iter->second.id);

fseek(foutput, -2, SEEK_CUR);
fputs("};\t// in decimal\n", foutput);
```

Fig. 4. Part of DBC2XML source code.

```
<template>
    <name> Transceiver </name>
    <parameter> const int num, const int &amp;id,
                const int &amp;period </parameter>
    <declaration>
        clock t;
        const int ArbTime = 1;
    ...
    </declaration>
    ...
    <transition>
        <source ref="id9"/>
        <target ref="id13"/>
        <label kind="assignment" x="-527"
               y="-110"> trans_vote++ </label>
        <nail x="-552" y="-119"/>
    </transition>
</template>
```

Fig. 5. Part of DBC2XML libraries.

If UPPAAL model is too complicated, a state explosion problem may occur or a verification time may become excessively long when performing model checking. To prevent this situation, DBC2XML generates an abstract UPPAAL model considering the following:

1) 25 μs is set to UPPAAL clock 1. It is preferable to set the nominal bit-time 2 μs to UPPAAL clock 1 when the bit rate of the CAN network is 500 kbps, but using a small unit of UPPAAL clock can lead to impractically long verification times.
2) A 3-bit IFS (Interframe space) field is included in the message frame.
3) All messages are transmitted by different ECUs, and aperiodic messages are not considered.

The UPPAAL model that DBC2XML automatically generated from DBC file of Fig. 2 is shown in the Fig. 6, and the comprehensive overview is shown in the Fig. 7. The UPPAAL model consists of one Bus model and several ECUs, and the ECU consists of one Application model and one Transceiver model.

The Bus model is shared by multiple ECUs transmitting messages. If there is no ECU to transmit a message, the Bus model is in an idle state, and when a ECU transmit s a message, the Bus model transitions to a busy state.

In the Application model, non-periodic messages are transitioned to the not_periodic state and are no longer considered. In the case of a periodic message, the Application model sets an offset as determined in the processing step, and generates a transmission signal of the message at fixed cycle time. Before the message transmission starts, the Application model transitions to the ready state and consumes time as much as jitter. Jitter includes the time the control logic is processed and the time the message waits in the transmit queue. In this paper, jitter is set to 100 μs as WCET (Worst Case Execution Time). When the transmission is completed, the Application model transitions to the idle state and waits for the next cycle.

The Transceiver model receives the transmission signal from the Application model and performs the arbitration process. The highest priority message among the messages participating in the arbitration process is transmitted successfully (transmitting state of Transceiver model in Fig. 6), and the other messages perform the arbitration process again for retransmission (failed state of Transceiver model in Fig. 6). Since the length of the arbitration field is 12 bits, 12 bit-time is consumed to perform the arbitration process (ArbTime). The message occupying the CAN bus through the arbitration process transmits the remaining bits of the frame, and the time corresponding to the number of bits is consumed (AfterArb).

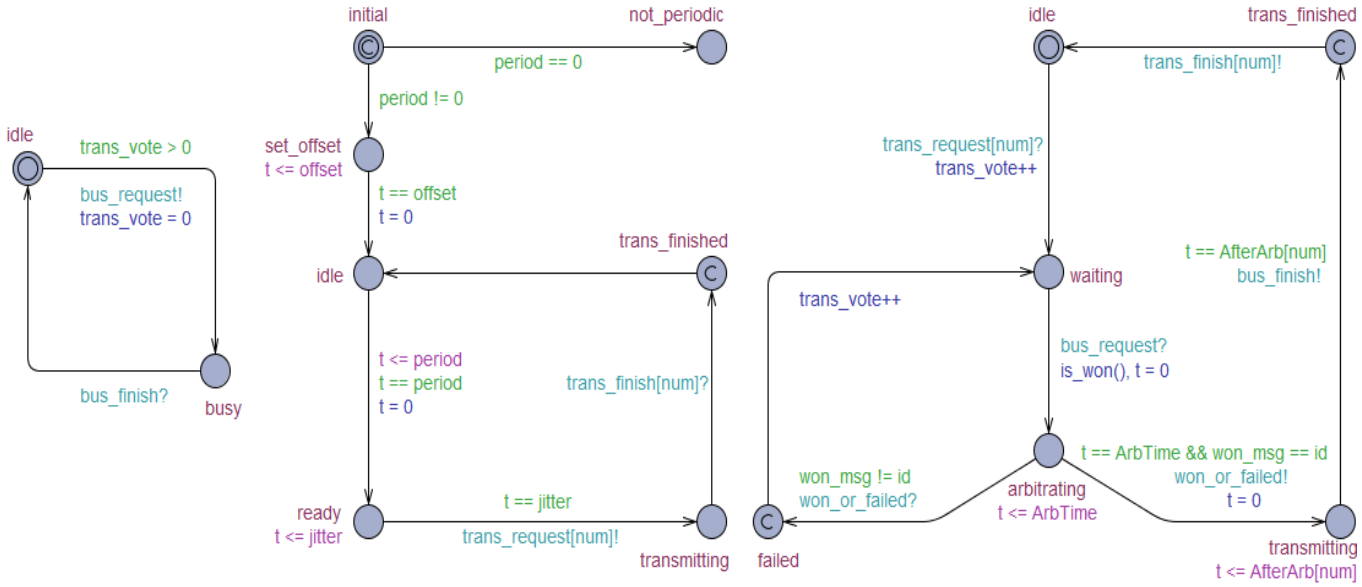

Fig. 6. Bus model, application model, and transceiver model.

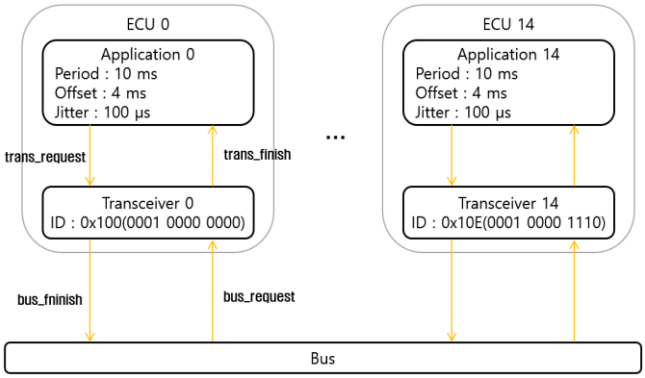

Fig. 7. Comprehensive overview of UPPAAL model.

C. UPPAAL

UPPAAL is a tool for modeling, simulating, and verifying a real-time system and specifies the behavior of the system with the network of automata with UPPAAL clock and variables [10]. Models specified through UPPAAL consist of templates, locations, edges, labels, etc. and are represented in XML format.

Model checking is performed by inputting the UPPAAL model automatically generated from DBC2XML and properties to be verified into UPPAAL. Model checking is a method to automatically explore all the state space of the model and check whether the properties are satisfied, and the properties are expressed in simplified TCTL (Timed Computation Tree Logic). If the UPPAAL model satisfies the property, the result is 'satisfied'. If the UPPAAL model does not satisfy the property, it returns 'not satisfied' with a counter example.

III. Verification with Model Checking

We verify the logical and timing properties of the model generated using DBC2XML. The verification was carried out on Intel Xeon E5-2680 2.7GHz with 256GB RAM, running Ubuntu 16.04 LTS.

A. Verifying Logical Properties of the Model

The UPPAAL model generated using DBC2XML should be formally verified to satisfy the logical properties of Table I. Properties can be expressed in simplified TCTL and we confirmed that the model satisfied all the logical properties as a result of performing model checking.

TABLE I: Logical Properties in Simplified TCTL

1. Two different messages should not be transmitted at the same time.	1. A[] not (tra0.transmitting and tra1.transmitting)
2. In the arbitration process, the situation that the higher priority message fails due to the lower priority message should not happen.	2. A[] not tra0.failed
3. If the message is being transmitted, the CAN bus must be in busy state.	3. A[] tra0.transmitting imply Bus.busy

B. Verifying Timing Properties of the Model

Through the proposed framework, we can verify the timing properties of a given DBC file. The given DBC file is as shown in the Fig. 2, assuming that the deadline for all messages is 1 ms as a requirement. That is, if the response time of the message is less than 1 ms, the message is successfully transmitted according to the requirement. If the response time exceeds 1 ms, the predetermined requirement is not satisfied and the transmission is delayed.

The results of the verification of the DBC file of the Fig. 2 using the proposed framework are shown in the Table II. The three messages with IDs 0x106, 0x109, and 0x10E were found to have the worst case response times of 1.025 ms, 1.250, and 1.650 ms, respectively. That is, the load on the CAN bus is very low at 28.5%, but the specific messages have been transmitted exceeding the deadline. If these messages were messages containing information directly related to the safety of the driver and the vehicle, delay of response time would have undermined the safety of the driver and the entire vehicle.

TABLE II: Result of Timing Properties Verification

Message ID	Cycle time (ms)	Offset (ms)	Response time (ms)
0x100	10	4	0.350
0x101	10	4	0.575
0x102	20	2	0.350
0x103	20	3	0.350
0x104	20	4	0.800
0x105	20	0	0.350
0x106	10	4	1.025
0x107	0	2	·
0x108	0	3	·
0x109	10	4	1.250
0x10A	5	0	0.525
0x10B	5	1	0.350
0x10C	5	2	0.575
0x10D	20	3	0.575
0x10E	10	4	1.650

IV. Conclusion

We propose a framework that can easily verify logical and timing properties using UPPAAL and DBC2XML which automatically generate UPPAAL model from CAN database. CAN network designers can use this framework to easily verify the safety of the CAN database at the vehicle design stage. It will also help reduce vehicle development costs.

However, there is a limitation that the CAN network is abstracted and modeled to prevent the state explosion problem and to minimize the verification time. In order to obtain more accurate results, the degree of abstraction should be lowered, and research on this will be left as future research.

References

[1] *ISO 11898-1: 2015 Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling.*

[2] K. Tindell and A. Burns, "Guaranteeing message latencies on Control Area Network (CAN)," in *Proc. the 1st International CAN Conference*, 1994.

[3] J. *Krakora* and Z. Hanzalek, "Timed automata approach to CAN verification," in *Proc. 11th IFAC Symposium on Information Control Problems in Manufacturing*, INCOM, Salvador, Elsevier, 2004.

[4] C. Pan, J. Guo, and L. F. Zhu, "Modeling and verification of CAN bus with application layer using UPPAAL," *Electronic Notes in Theoretical Computer Science*, vol. 309, pp. 31-49, December 22, 2014.
[5] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis, "Controller area network (CAN): Response time analysis with offsets," in *Proc. the 9th IEEE International Workshop on Factory Communication System*, 2012, pp. 43-52.
[6] S. Mubeen, J. M. Turja, and M. Sjodin, "Extending offset-based response-time *analysis* for mixed messages in controller area network," presented at 18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Cagliari, Italy, Sep. 10-13, 2013.
[7] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183-235, April 25, 1994.
[8] J. P. Katoen, *Concepts, Algorithms, and Tools for Model Checking*, pp. 189-256, 1999, ch. 4.
[9] E. M. Clarke, "The birth of model checking," *25 Years of Model Checking*, Springer, pp. 1-26, 2008.
[10] G. Behrmann, A. David, and K. G. Larsen. (2006). A Tutorial on Uppaal 4.0. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9120

Beomyeon Cho was born in Seongnam, Korea on February 11, 1989. He received the BS degree in computer science from Korea University in 2016. He started MS degree in electronic control from March 2016.

His research interests are formal methods and automotive communication protocols (CAN, automotive ethernet, and WAVE).

Taewook Kim was born in Seoul, Korea on September 22, 1990. He received the BS degree in electronic engineering from Kookmin University. He started MS degree in electronic control from March 2016.

His research interests include formal methods, software integration and safety.

Jin-Young Choi was born in 1959 in Korea. He received the BS degree in computer engineering from Seoul National University in 1982. He received MS degree in computer science, Drexel University in 1986 and acquired the Ph. D degree in computer science from the University of Pennsylvania in 1993. His research interests are real-time computing, formal methods (formal specification, formal verification, and model checking), process algebras, SDN, security, and software engineering.

He is a professor in Graduate School of Information Security in Korea University.

Virtual Reality and Technology