Data Platform for the Connection of Cognitive Ports

Title:	Document Version:
D3.5 Data processing services M27	1.1

Project Number:	Project Acronym:	Project Title:
H2020-871493	DataPorts	A Data Platform for the Cognitive Ports of the Future

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M27 (March 2022)	M28 (April 2022)	O- PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other; ORDP: Open Research Data Pilot; E: Ethics.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Andreu Belsa	UPV	WP3

Authors (organisation):	
Andreu Belsa (UPV)	Anastasios Nikolakopoulos (ICCS)
Matilde Julián (UPV)	Achilleas Marinakis (ICCS)
Carlos E. Palau (UPV)	Vrettos Moulos (ICCS)
José Antonio Clemente (PRO)	George Magafossis (ICCS)
Héctor Iturria (PRO)	Xhulja Shahini (UDE)
Miguel Bravo (ITI)	Tristan Kley (UDE)
Santiago Cáceres (ITI)	Pablo Giménez (VPF)

Abstract:

This deliverable describes the implementation of the Semantic Interoperability Component and the Data Abstraction and Virtualization Component, which constitute the Data Processing Services of the DataPorts Platform. For each of these components, this document provides a summary of the functionality of the component and its relationships with other components of the platform, a technological description of the component, which covers its implementation and main technological challenges, a summary of the current implementation status, including information about its official release, and the plans, as well as an example of use.

Keywords:

Semantic Interoperability, Data Interoperability, Data Models, Orion Context Broker, Fiware, Ontologies, Metadata Interoperability, Virtual Data Container, Virtual Data Repository, Data Abstraction and Virtualization

Revision History

Revision	Date	Description	Author (Organisation)
V0.1	07/02/2022	First version of the document	Andreu Belsa (UPV), Matilde Julián (UPV) and Carlos E. Palau (UPV)
V0.2	02/03/2022	Contributions from UPV, PRO and ICCS. Sections 1,3,4,5 and 6 updated.	Andreu Belsa (UPV), Matilde Julián (UPV), José Antonio Clemente (PRO), Héctor Iturria (PRO), Anastasios Nikolakopoulos (ICCS), Achilleas Marinakis (ICCS), Vrettos Moulos (ICCS) and George Magafossis (ICCS)
V0.3	11/03/2022	Added contributions from UPV, PRO, ICCS, ITI, UDE and VPF to Sections 3 and 7. Section 8 added. Section 9 updated.	Andreu Belsa (UPV), Matilde Julián (UPV), José Antonio Clemente (PRO), Héctor Iturria (PRO), Anastasios Nikolakopoulos (ICCS), Achilleas Marinakis (ICCS), Vrettos Moulos (ICCS), George Magafossis (ICCS), Miguel Bravo (ITI), Xhulja Shahini (UDE), Tristan Kley (UDE) and Pablo Giménez (VPF)
V1.0	14/03/2022	Section 10 updated. Pre-submission check performed.	Andreu Belsa (UPV)), Matilde Julián (UPV) and Carlos E. Palau (UPV)
V1.1	25/03/2022	Update after review by ITI, EVR and UniKoblenz	Andreu Belsa (UPV), Matilde Julián (UPV), Santiago Cáceres (ITI), José Antonio Clemente (PRO) and Achilleas Marinakis (ICCS).

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement № 871493.

More information available at https://DataPorts-project.eu

Copyright Statement

INDEX

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 DATAPORTS PROJECT OVERVIEW

DataPorts is a project funded by the European Commission as part of the H2020 Big Data Value PPP programme and coordinated by the ITI - Technological Institute of Informatics. DataPorts relies on the participation of 13 partners from five different nationalities. The project involves the design and implementation of a data platform, its deployment in two relevant European seaports connecting to their existing digital infrastructures and addressing specific local constraints. Furthermore, a global use case involving these two ports and other actors and targeting inter-port objectives, and all the actions to foster the adoption of the platform at European level.

Hundreds of different European seaports collaborate with each other, exchanging different digital data from several data sources. However, to achieve efficient collaboration and benefit from AI-based technology, a new integrating environment is needed. To this end, DataPorts project is designing and implementing an Industrial Data Platform.

The DataPorts Platform aim is to connect to the different digital infrastructures currently existing in digital seaports, enabling the interconnection of a wide variety of systems into a tightly integrated ecosystem. In addition, it intends to set the policies for a trusted and reliable data sharing and trading based on data owners' rules and offering a clear value proposition. Finally, it also strives to leverage on the data collected to provide advanced Data Analytic services based on which the different actors in the port value chain could develop novel AI and cognitive applications.

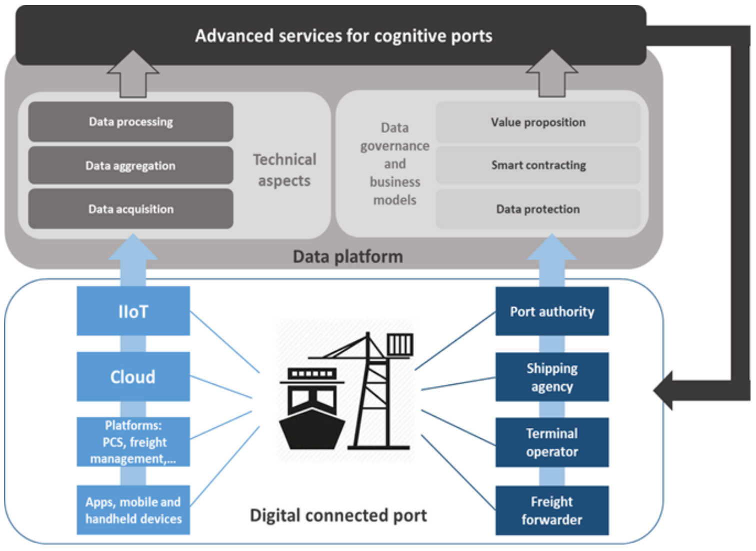


Figure 1 - DataPorts project overview

DataPorts will allow establishing a future Data Space unique for all maritime ports of Europe and contribute to the EC global objective of creating a Common European Data Space.

1.2 DELIVERABLE PURPOSE AND SCOPE

Specifically, the DOA states the following regarding this Deliverable:

"This deliverable will involve the services offered by the platform in the different steps of the data value chain. It includes the abstraction and virtualization of the collected data, and data semantic interoperability".

The purpose of this document is to describe the tools and interfaces implemented to provide semantic interoperability in the DataPorts Platform, as well as the implementation of the services offered by the platform for data management and virtualization.

1.3 DELIVERABLE CONTEXT

Its relationship to other documents is as follows:

Primary Preceding documents:

- D2.1 Industrial Data Platforms and seaport community requirements and challenges: Provides the technical requirements of the DataPorts Platform.
- D2.2 Scalability, Interoperability and Definition Standards: The process of data modelling involves technical partners working closely with business stakeholders, as well as potential users of the information system. Both the M18 and M27 versions of this deliverable present the status of this joint task from the technical point of view. However, the other perspectives of this joint work are reflected in deliverable D2.2.
- D2.4 Platform architecture and specifications: Provides the architecture and specifications of the DataPorts Platform.
- D3.1 Data Access Interfaces: The Semantic Interoperability component has significant interactions with the Data Access component and vice versa. Therefore, it is recommended reading both the D3.1 and D3.5 to capture the complete picture of the joint work between these two components.
- D3.2 Data Processing Services M27: Presents the initial version of the services described in this deliverable.
- D5.3 Use Case Oriented Pilots Initial Version M24: It provides an overview of the usage of the different components of the platform in the scenarios implemented in the project demonstration. For each of the scenarios is described which DataPorts components are used, and which are their interactions.

1.4 DOCUMENT STRUCTURE

This deliverable is broken down in the following sections:

- **Section 1 Introduction**: It includes an overview of the project, a short description of the purpose and scope of this document and the dependencies with other deliverables.
- **Section 2 Technical objectives:** It explains the technical objectives that each of the components described in this deliverable fulfils according to the project proposal.
- **Section 3 Position in the DataPorts Architecture:** It briefly describes the relationships that each of the components presented in this deliverable has with the rest of the components in the architecture.
- **Section 4 Data Access Component Improvements (M18-M27)**: it provides an update of the Data Access Component implementation described in deliverable D3.1 (M18).
- **Section 5 Semantic Interoperability Component**: It describes in detail the Semantic Interoperability Component. The description includes an overview of the component, its technological description and the status and roadmap the component, as well as an example of use in the context of the DataPorts Platform.
- **Section 6 Data Abstraction and Virtualization Component:** It describes in detail the Data Abstraction and Virtualization Component. The description includes an overview of the component, its technological description and the status and roadmap the component, as well as an example of use in the context of the DataPorts Platform.
- **Section 7 Data Processing Services common example of use:** it presents an example of use of the platform, which shows the integration of the data processing services and relevant analytics services.
- **Section 8 Release summary:** It provides information of the release of each component described in this deliverable, including component name, version, source code, documentation, Docker image and dependencies.
- **Section 9 Conclusions:** It contains the conclusions regarding the components described in this deliverable.

Annexes:

- **Annex A Metadata Registry:** It describes the internal data model of the Metadata Registry subcomponent.

1.5 DOCUMENT DEPENDENCIES

This document is part of an iteration of living deliverables. The previous version of the document, deliverable D3.2 Data processing services M18, describes the first iteration of the Data Processing Services offered by the platform in the different steps of the data value chain. The final version of those services is presented in this document.

The main sections updated in D3.5 from D3.2 are:

- Section 1 updates the general information of the deliverable to the current period of delivery (M27).
- Section 2 has no changes compared to D3.2.
- Section 3 includes new sequence diagrams describing the data flows between the DataPorts Platform components with more details.
- Section 4 updates the content of D3.1. This new information is important to understand the way to retrieve and exchange data of the components described in D3.5 and D3.6.
- Section 5 describes the current version of the Semantic interoperability component and the DataPorts Data Model including all progress made since M18.
- Section 6 describes the current version of the Data Abstraction and Virtualization component including all progress made since M18.
- Section 7 offers a new section focused in to show the functional operation of the Data Processing Services with real data sources and the interaction between the Data Processing Services and the components described in D3.1 and D3.6.
- Section 8 provides a new section with the technical release information of the components involved in this deliverable.
- Section 9 updates the deliverable conclusions to the current period of delivery (M27).

D3.5 is delivered in the same period as D3.6 and D3.7. All these deliverables together with the D3.1 (delivered in the M18) are providing the final design of each component of the platform, mainly including a technical description and a complete release summary of each component of the platform.

The final version of the DataPorts Platform implementation, according to milestone number 11 of the project, will be available in M30. The section 8 of this deliverable provides the information about how this material will be released.

2 TECHNICAL OBJECTIVES

This deliverable describes the implementation of the components called “Data Semantic Interoperability” and “Data abstraction and virtualization”. The lack of interoperability among data is hampering the real adoption of data driven solutions in the seaports [1]. There are several challenges related with the interoperability and integration of data collected from multiple sources that must be addressed to take real advantage of the existing technical infrastructure and the availability of the huge amount of data generated by the different stakeholders. Those entail the technical challenges of DataPorts project to make real the transition of seaports from digital and connected to smart and cognitive. They are summarised in the following technical objectives of the project:

- *O1. “To address real-life data market use cases in two relevant European seaports, two global use cases and related communities”:* The rationale of the components presented in this deliverable is that the current solutions underutilise (or do not utilise) the existing tools for acquisition, aggregation, processing, and analysis of the data coming from the different stakeholders, sources and existing platforms.
- *O2. “To design and validate next-generation set of advanced interoperable data related and AI based services”:* The aim of the components presented in this deliverable is to provide the data acquisition mechanisms, the data sanitisation algorithms to guarantee data quality; the ability of federating data varying in syntax and semantics; and efficient and effective techniques for data wrapping that will represent the underlying mechanism for supporting selective release, storage and analytics on data enabling efficient processing over protected data while preventing (or limiting) access to the actual data content by other parties.
- *O3. “To define an engineering methodology facilitating application of DataPorts architecture and tools, to support cognitive, privacy-aware and secure data sharing processes”:* The aim of the components involved in this deliverable is to deliver new services and components of the DataPorts Platform based on the requirements, functionalities and the architecture designed in WP2.
- *O4. “To define, design and incorporate a novel, scalable, resilient, semantic approach for data sharing”:* The aim of the components involved in this deliverable is to provide interoperable and semantic solutions and mechanisms offering support to the Big Data Analytics components of the platform.

To sum up, the deliverable presents the services offered by the DataPorts Platform that enable the semantic interoperability and the abstraction and virtualization of the collected data from the different digital infrastructures currently existing in digital seaports. Those services enable the integration of a wide variety of systems into a tightly integrated ecosystem and leverage the data collected to be used by advanced Data Analytics services.

The Semantic Interoperability Component fulfils the following technical objectives of the WP3:

- *O3.2. “To define ontologies, mechanisms and enablers to provide semantic interoperability with data platforms, IoT devices, robots and other data sources, and develop the semantic-based tools needed to facilitate generation of interfaces for sensing and actuation required in the DataPorts data platform”*.
- *O3.3. “To define mechanisms for data management, virtualization including streaming techniques”.* More concretely, the semantic interoperability is focused on facilitating the data management of the data sources integrated in the DataPorts data platform.

In addition, the Semantic Interoperability Component fulfils the following technical objectives defined by the task description in the GA:

- Offering a framework for semantic interoperability of diverse data platforms, autonomous systems, and applications.

- Providing business level interoperability using existing ontologies and developing a semantic framework for describing ports data together with mappings to standard vocabularies to simplify the reuse of data apps for analytics and forecasting. This semantic framework will codify domain knowledge of the domain experts, and thus can be reused and exploited by the data experts directly, thereby empowering building cognitive port applications.
- Defining and implementing the core mechanisms enabling the deployment of highly distributed Data Spaces, considering context components.
- Analysing and implementing software mechanisms for retrieving, aggregating, and facilitating the connection with industrial data spaces from external platform.

The Data Abstraction and Virtualization Component fulfils the following technical objectives of the WP3:

- O3.1 "*To identify different data sources to be integrated in the DataPorts data platform, including the mechanisms to store and facilitate data management*".
- O3.3 "*To define mechanisms for data management, virtualization including streaming techniques*".

More concretely, the component aims at implementing an abstraction and virtualization mechanism for data coming from a variety of diverse sources, supporting Big Data repositories and technologies that can use structured or non-structured data, relational or NoSQL databases as well as web services and APIs.

In addition, the Data Abstraction and Virtualization Component fulfils the following technical objectives defined by the task description in the GA:

- Developing a framework for accessing data coming from different and heterogeneous data sources.
- Implementing the Virtual Data Container (VDC) concept.
- Assisting the application developers to focus only on the required data, and letting the proposed framework to deliver them, thus encapsulating all the underlying technical complexity.

3 POSITION IN THE DATAPORTS ARCHITECTURE

From the services' perspective of the DataPorts Platform, the "Data Semantic Interoperability" and "Data Abstraction and Virtualization" are located inside the "Data Processing Services" building block (see Figure 2). This building block also includes the "Data Access Component" (explained in D3.1 [2]), which enables access to the available data sources. The "Data Processing Services" are exchanging data and metadata with the "Data Governance Services" described in D3.4 [3] and D3.7 [4], and with the "Analytics Services" described in D3.6 [5].

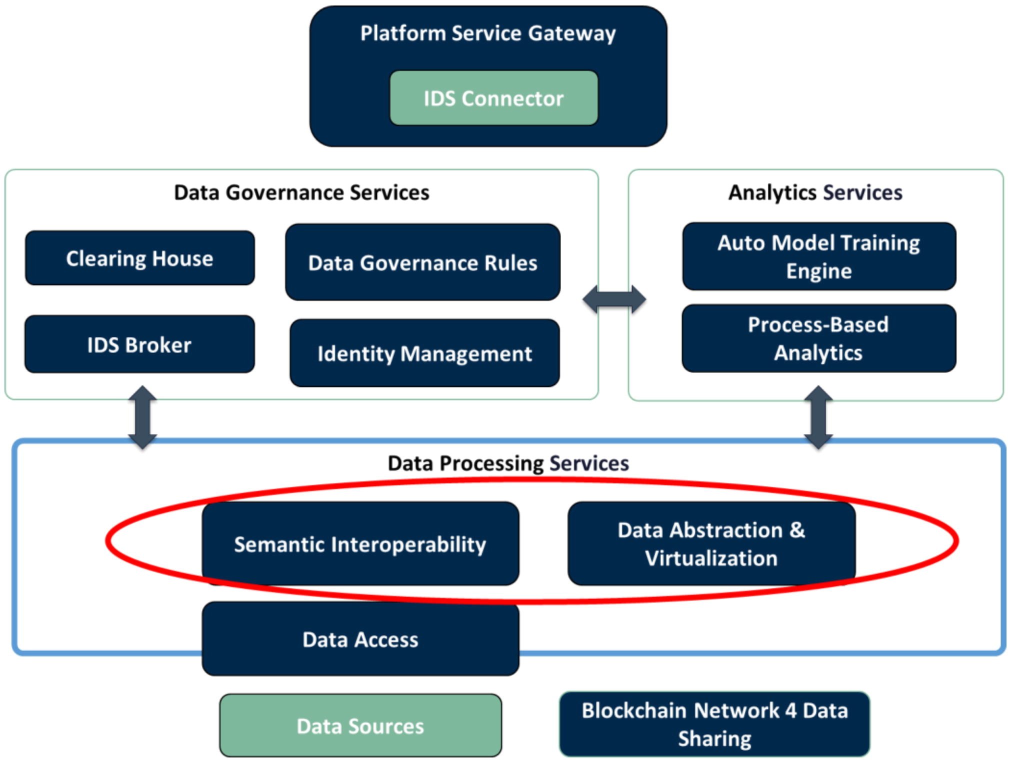


Figure 2 – DataPorts Platform building blocks

The position of the Semantic Interoperability Component in the DataPorts architecture, which is described in detail in D2.4 [6], is shown in Figure 3. The Semantic Interoperability Component acts as a middleware layer between the Data Access component and the upper layers (Data Abstraction and Virtualization component, Automatic Model Training Engine and Process-Based Analytics). Hence, in the architecture, the Semantic Interoperability Component is on top of the Data Access Component and below the components that obtain the data from its API, such as the Data Abstraction and Virtualization component, the Automatic Model Training Engine and the Process-Based Analytics component (Figure 2). The API of this component offers access to the available data and metadata. There are two modes of access to the data: the first one is based on the use of subscriptions to receive near real-time data, while the second one enables queries on demand to receive historical data or near real-time data. Moreover, the Semantic Interoperability Component provides a description and a repository for the DataPorts Data Model and DataPorts Ontology. In addition, the Semantic Interoperability Component is connected to the Data Governance component to exchange data source metadata and be able to check the user's permissions.

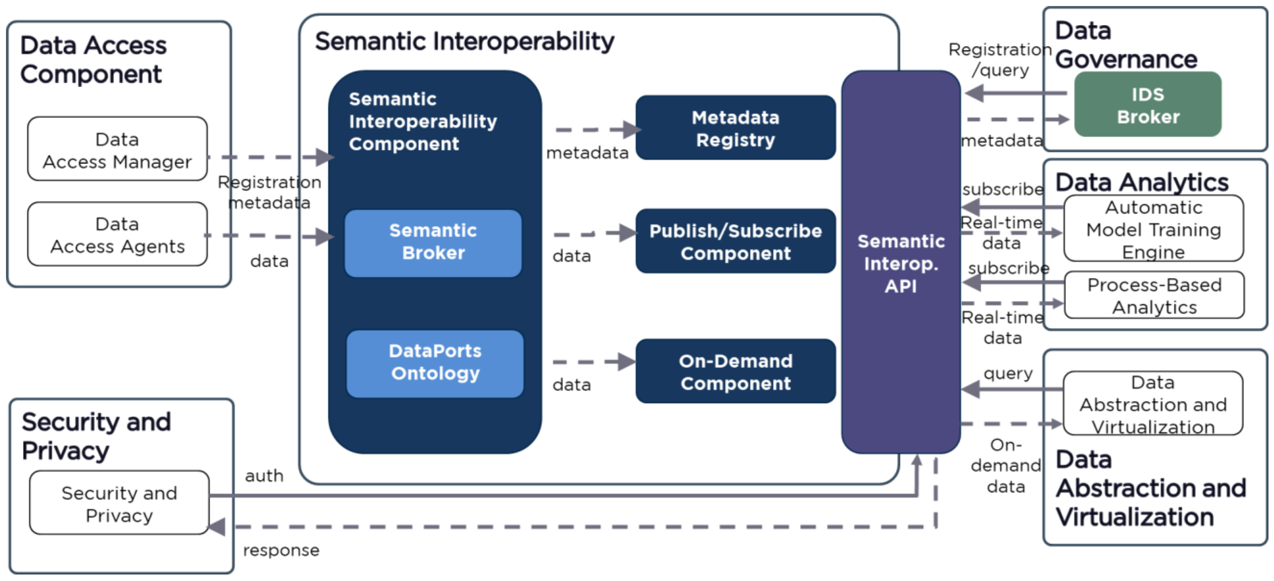


Figure 3 - Interactions of Semantic Interoperability Component

The Data Abstraction & Virtualization (DAV) component receives data from the Data Access and Semantic Interoperability layers, as well as any possible data provider that resides outside these two components. On the other hand, DAV provides data to the analytics components. To summarise, DAV serves as a link between DataPorts principal data providers and consumers, by gaining persistent data input (mostly from the Semantic Interoperability layer) and forwarding these streams to any interested recipient. The position of DAV with respect to other components of the DataPorts architecture is shown in Figure 4. The internal structure of DAV as well as its functionalities, which are also depicted in that figure, are analysed in Section 6.

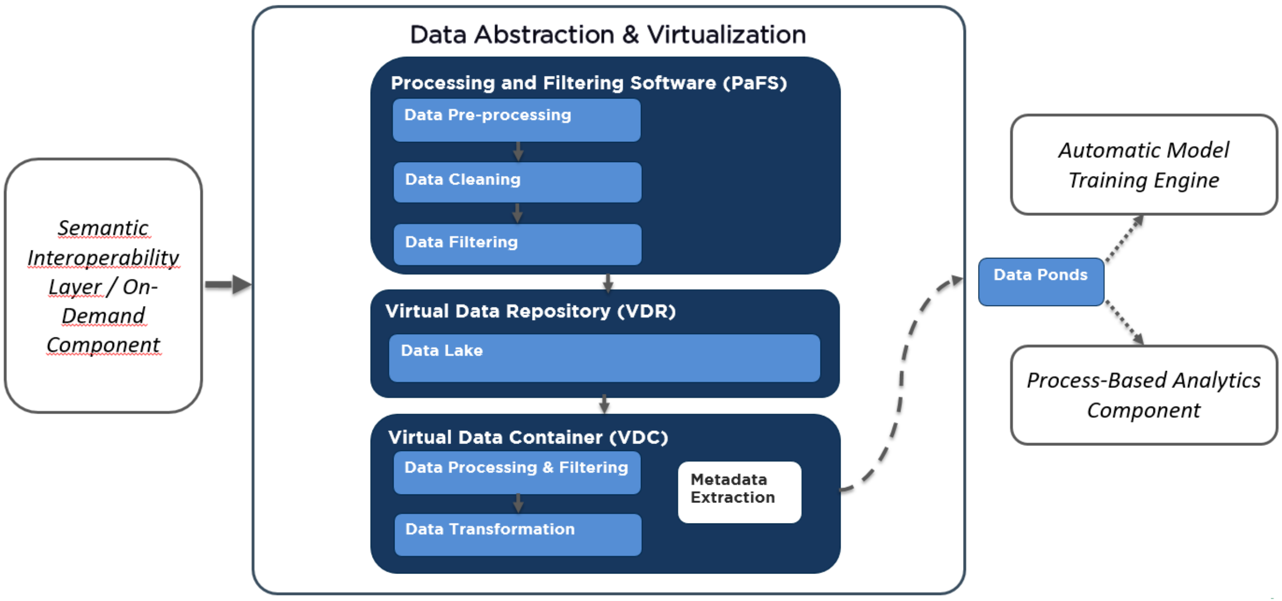

Semantic Interoperability Layer / On-Demand Component
Data Abstraction & Virtualization
Processing and Filtering Software (PaFS)
Data Pre-processing
Data Cleaning
Data Filtering
Virtual Data Repository (VDR)
Data Lake
Virtual Data Container (VDC)
Data Processing & Filtering
Data Transformation
Metadata Extraction
Data Ponds
Automatic Model Training Engine
Process-Based Analytics Component

Figure 4 - Interactions of Data Abstraction and Virtualization component

Regarding the details of the integrations described above, the data flows for typical scenarios involving data provider and data consumers have been defined to illustrate how the components interact with each other. The first of these flows describes the process of connecting a new data source to the platform. According to this flow, when a new data source is made available (Figure 5), the user creates the corresponding agent in the Data Access Component and defines the data governance metadata in the Data Governance component. Next, the metadata of the data source and its corresponding agent (or agents) is stored locally in the Semantic

Interoperability component. Next, the Semantic Interoperability component sends the data source metadata to the Data Governance (through a subscription), where it is combined with the data governance metadata regarding the access permissions provided by the user. The Data Governance component and its integration with Semantic Interoperability, as well as the role of the Identity Manager shown in Figure 5, are described in detail in deliverable D3.7 [4].

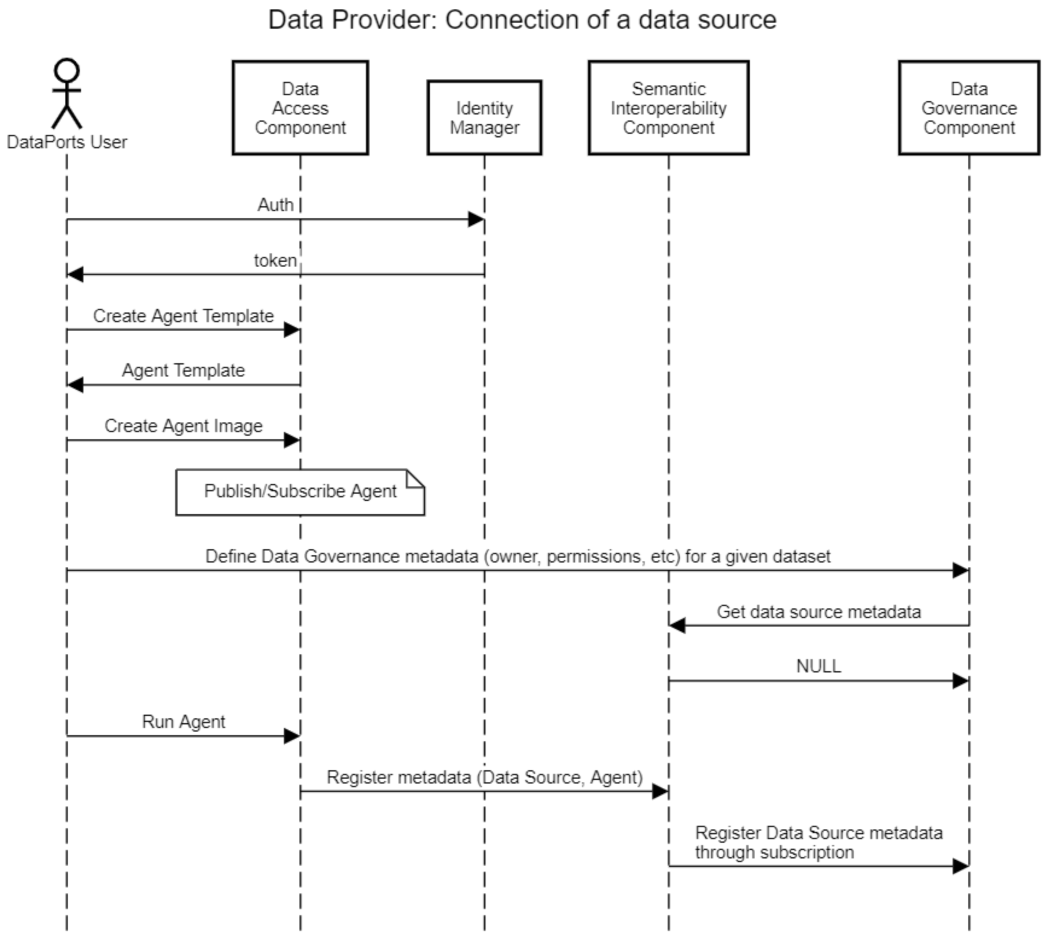

Data Provider: Connection of a data source
DataPorts User
Data Access Component
Identity Manager
Semantic Interoperability Component
Data Governance Component
Auth
token
Create Agent Template
Agent Template
Create Agent Image
Publish/Subscribe Agent
Define Data Governance metadata (owner, permissions, etc) for a given dataset
Get data source metadata
NULL
Run Agent
Register metadata (Data Source, Agent)
Register Data Source metadata through subscription

Figure 5 – Connection of a new data source

Once the data source has been registered, the data is available for the upper layers (provided that the user has the proper permissions). One of such components is DAV component, which obtains the historical data from the Semantic Interoperability component and the Data Access component (Figure 6), pre-processes it and puts the result in a temporary storage to make it available for the Data Analytics services. More details of the integration of the DAV component with the Semantic Interoperability component and the Data Access component are explained in Section 4.4.

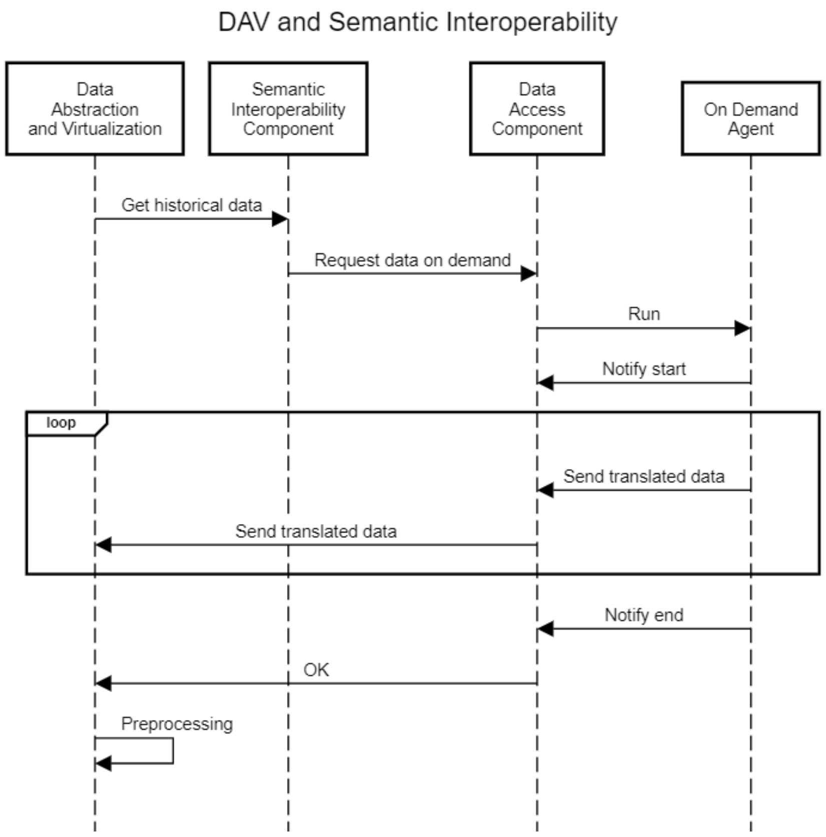


Figure 6 – Integration of the Semantic Interoperability component with the Data Access component and Data Abstraction and Virtualization

The components that provide AI services, such as the Automatic Model Training Engine or the Process-Based Analytics component, can request a pre-processed dataset to the DAV component to train a model (Figure 7 and Figure 8). Finally, the trained model will be able to either perform predictions based on the historical data or call the Semantic Interoperability component to request the current values that will be processed to provide a result. The Automatic Model Training Engine and the Process-Based Analytics component are described in detail in deliverable D3.6 [5]. Moreover, Section 7 presents an example of how the integration between the Data Processing Services and the Data Analytics Services would work in a scenario of the pilots.

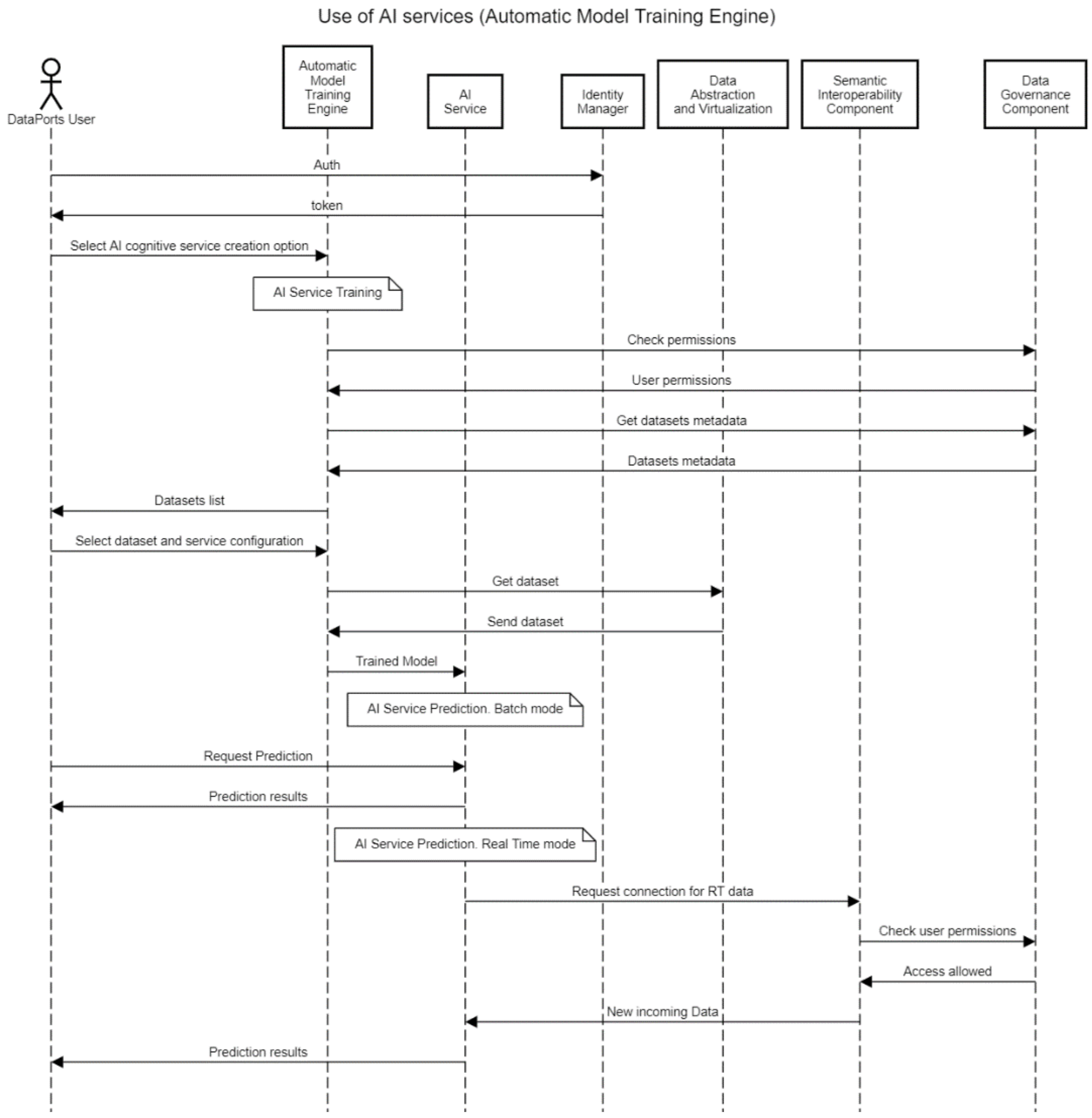

Use of AI services (Automatic Model Training Engine)
DataPorts User
Automatic Model Training Engine
AI Service
Identity Manager
Data Abstraction and Virtualization
Semantic Interoperability Component
Data Governance Component
Auth
token
Select AI cognitive service creation option
AI Service Training
Check permissions
User permissions
Get datasets metadata
Datasets metadata
Datasets list
Select dataset and service configuration
Get dataset
Send dataset
Trained Model
AI Service Prediction. Batch mode
Request Prediction
Prediction results
AI Service Prediction. Real Time mode
Request connection for RT data
Check user permissions
Access allowed
New incoming Data
Prediction results

Figure 7 – Integration of the Semantic Interoperability component and the Data Abstraction and Virtualization with the Automatic Model Training Engine

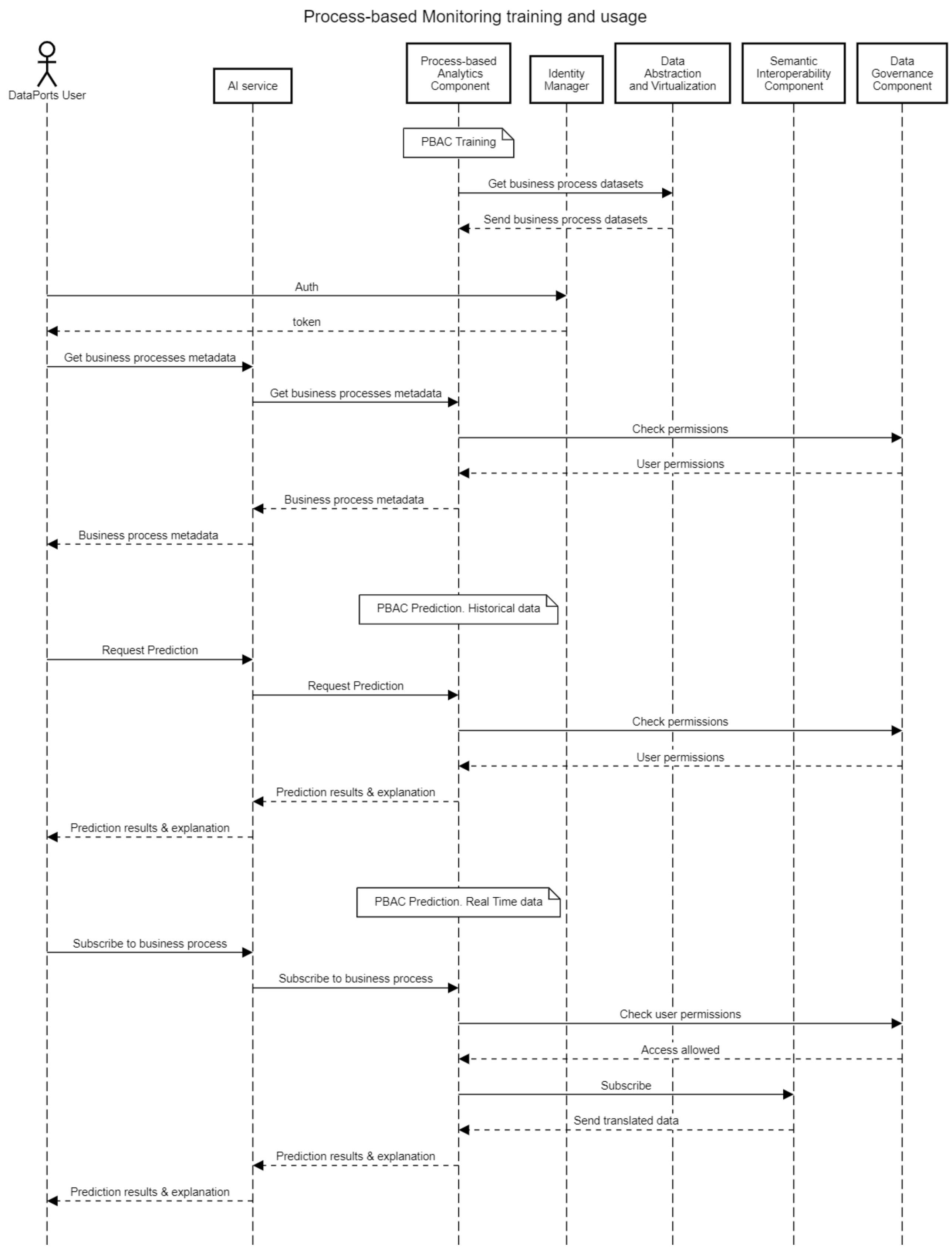

Process-based Monitoring training and usage
DataPorts User
AI service
Process-based Analytics Component
Identity Manager
Data Abstraction and Virtualization
Semantic Interoperability Component
Data Governance Component
PBAC Training
Get business process datasets
Send business process datasets
Auth
token
Get business processes metadata
Get business processes metadata
Check permissions
User permissions
Business process metadata
Business process metadata
PBAC Prediction. Historical data
Request Prediction
Request Prediction
Check permissions
User permissions
Prediction results & explanation
Prediction results & explanation
PBAC Prediction. Real Time data
Subscribe to business process
Subscribe to business process
Check user permissions
Access allowed
Subscribe
Send translated data
Prediction results & explanation
Prediction results & explanation

Figure 8 – Integration of the Semantic Interoperability component and the Data Abstraction and Virtualization with the Process-Based Analytics component

4 DATA ACCESS COMPONENT IMPROVEMENTS (M18-M27)

The development of Data Access Component (T3.1) was completed in M18. However, several improvements have been developed in recent months (M18 – M27). These improvements were not only aimed at improving or completing the component, but also at facilitating integration with the higher layers (DAV).

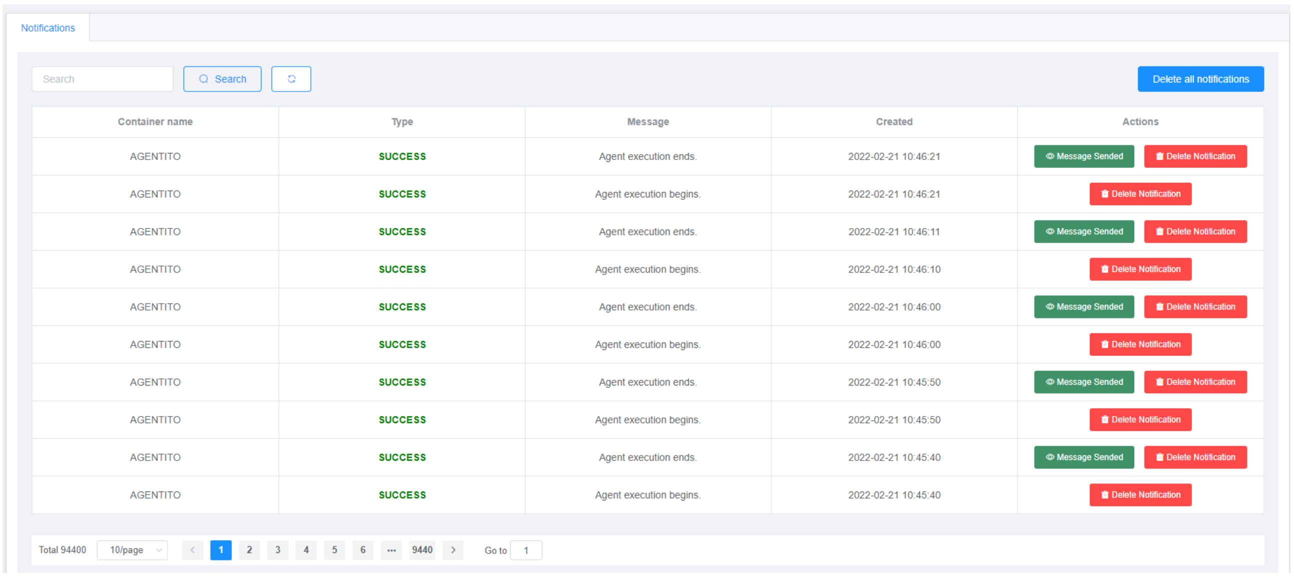
These improvements include:

- Agent notifications
- Docker containers management for On-demand agents
- New templates for the component
- Use of Cygnus for integration with Data Abstraction and Virtualization

These improvements will be explained in the following subsections.

4.1 AGENTS NOTIFICATION

A mechanism to overview the status of the agents has been implemented. This mechanism is based on the use of notifications that the agents send to the Data Access Manager to inform of the actions performed (start, send data, finish) or if an error occurs. This mechanism makes use of an API developed on the DAC server. It receives the actions that are being executed on the agents. Each time a notification is received; it is sent to the UI via web socket and is displayed in the correct layout. To display this information, the Log view has been developed. Next picture depicts that view, where it is possible to see the notifications received the container they are associated with and the type of action they correspond to.

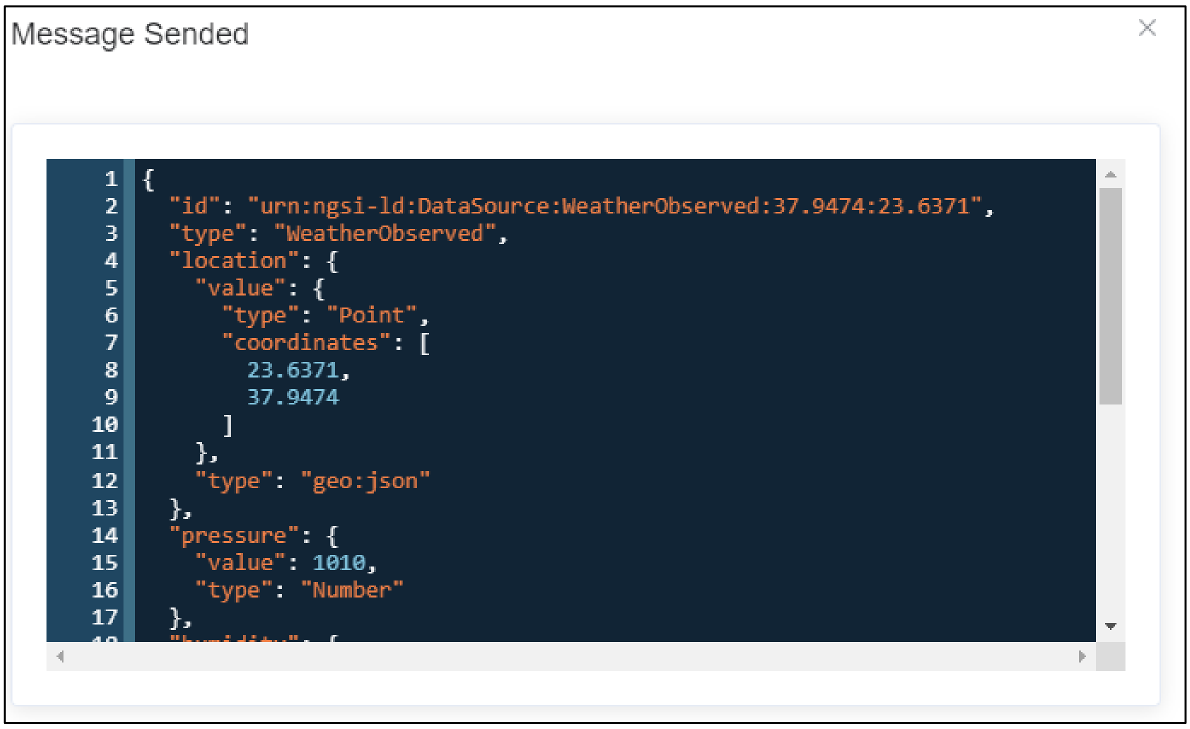


Container name	Type	Message	Created	Actions
AGENTITO	SUCCESS	Agent execution ends.	2022-02-21 10:46:21	Message Sended, Delete Notification
AGENTITO	SUCCESS	Agent execution begins.	2022-02-21 10:46:21	Delete Notification
AGENTITO	SUCCESS	Agent execution ends.	2022-02-21 10:46:11	Message Sended, Delete Notification
AGENTITO	SUCCESS	Agent execution begins.	2022-02-21 10:46:10	Delete Notification
AGENTITO	SUCCESS	Agent execution ends.	2022-02-21 10:46:00	Message Sended, Delete Notification
AGENTITO	SUCCESS	Agent execution begins.	2022-02-21 10:46:00	Delete Notification
AGENTITO	SUCCESS	Agent execution ends.	2022-02-21 10:45:50	Message Sended, Delete Notification
AGENTITO	SUCCESS	Agent execution begins.	2022-02-21 10:45:50	Delete Notification
AGENTITO	SUCCESS	Agent execution ends.	2022-02-21 10:45:40	Message Sended, Delete Notification
AGENTITO	SUCCESS	Agent execution begins.	2022-02-21 10:45:40	Delete Notification

Figure 9 – Notifications received from the agents

It is even possible to see the message sent to ORION (in case of a publish-subscribe agent) or to Cygnus[1] (in case of On-demand agent). By clicking on the Message sent button, it is possible to see the content of the message, as shown in the following figure.

[1] https://fiware-cygnus.readthedocs.io/en/latest/



Message Sended

```
{
  "id": "urn:ngsi-ld:DataSource:WeatherObserved:37.9474:23.6371",
  "type": "WeatherObserved",
  "location": {
    "value": {
      "type": "Point",
      "coordinates": [
        23.6371,
        37.9474
      ]
    },
    "type": "geo:json"
  },
  "pressure": {
    "value": 1010,
    "type": "Number"
  },
```

Figure 10 – Example of message sent by an agent

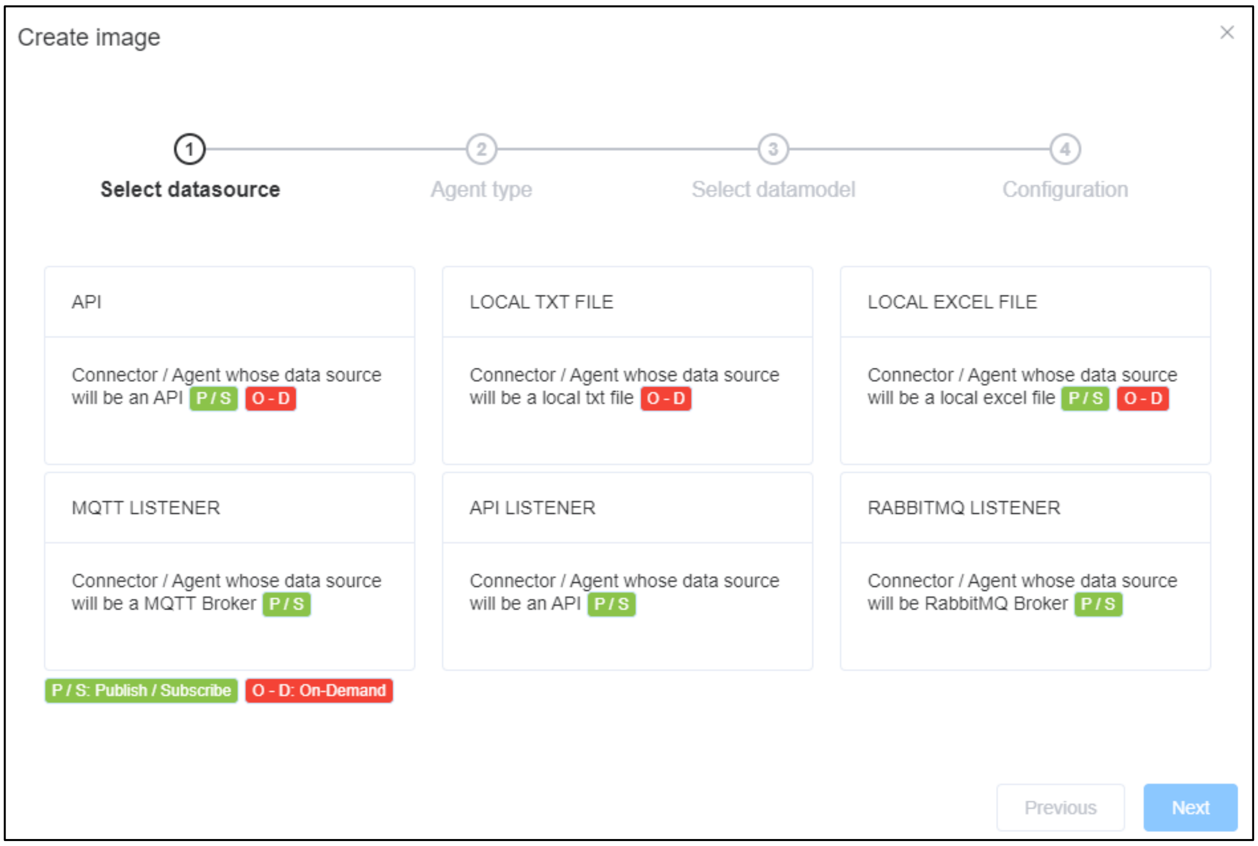
4.2 DOCKER CONTAINERS MANAGEMENT FOR ON-DEMAND CONTAINERS

On-demand agents are designed for large volumes of data. Therefore, they are usually executed only once. After this, the container must be deleted.

In the first version of DAC, the management of these containers was not done automatically. There was a process in background that checked every 10 minutes if an on-demand agent had finished and was still running. If so, the container was deleted. The agent management functionality has been improved in the current version of the component, in which the agent itself oversees its execution and sends notifications to the Data Access Manager to announce when the execution starts and when it ends or if an error occurs. Once the agent has been executed, the agent itself is in charge of ending its execution. This mechanism is more efficient than the previous one because it reduces the consumption of resources by keeping processes running once they have finished.

4.3 NEW TEMPLATES FOR THE DAC

The DAC has a wizard with a series of templates to facilitate the creation of agents. This suite of templates must be aligned with the development of the agents (WP5). Therefore, the number of templates will increase according to the needs of the agents to be developed. Since the delivery of the component (M18) until now, more templates have been included in the wizard. The list of available templates can be seen in the following image.


Create image

1 Select datasource — 2 Agent type — 3 Select datamodel — 4 Configuration

API	LOCAL TXT FILE	LOCAL EXCEL FILE
Connector / Agent whose data source will be an API P / S O - D	Connector / Agent whose data source will be a local txt file O - D	Connector / Agent whose data source will be a local excel file P / S O - D
MQTT LISTENER	API LISTENER	RABBITMQ LISTENER
Connector / Agent whose data source will be a MQTT Broker P / S	Connector / Agent whose data source will be an API P / S	Connector / Agent whose data source will be RabbitMQ Broker P / S

P / S: Publish / Subscribe O - D: On-Demand

Figure 11 – Agent templates

The templates shown in the image above are for:

- API. Creates an agent that connects to an API. This type of template allows the creation of both types of agents (publish/subscribe and on-demand).
- LOCAL TXT FILE. Creates an agent that reads the content of a local txt file. This agent is an on-demand agent.
- LOCAL EXCEL FILE. Creates an agent that reads the content of a local excel file. This agent could be publish/subscribe and on-demand.
- MQTT LISTENER. This agent is a listener. So, it is an agent that is waiting data from a MQTT server. The template creates a publish/subscribe agent.
- API LISTENER. This agent is a listener. In that case, reads data from an endpoint. The template creates a publish/subscribe agent.
- RABBITMQ LISTENER. This agent is also a listener. The agent is waiting data coming from a RABBITMQ server. This is a publish/subscribe agent.

4.4 USE OF CYGNUS FOR THE INTEGRATION WITH DAV

On-demand agents are not integrated with ORION. Their data are sent to the DAV. When dealing with large volumes of data it was very common to get the following error: ‘Request entity too large’. Therefore, the data is sent divided into blocks of 100 entities. This resulted in having the data at the destination divided into blocks. Some mechanism has to be devised to re-integrate those blocks and have the complete entity.

Therefore, it was decided to make use of Fiware Cygnus, which is based on Apache Flume[2], to reassemble the historical data (Figure 12).

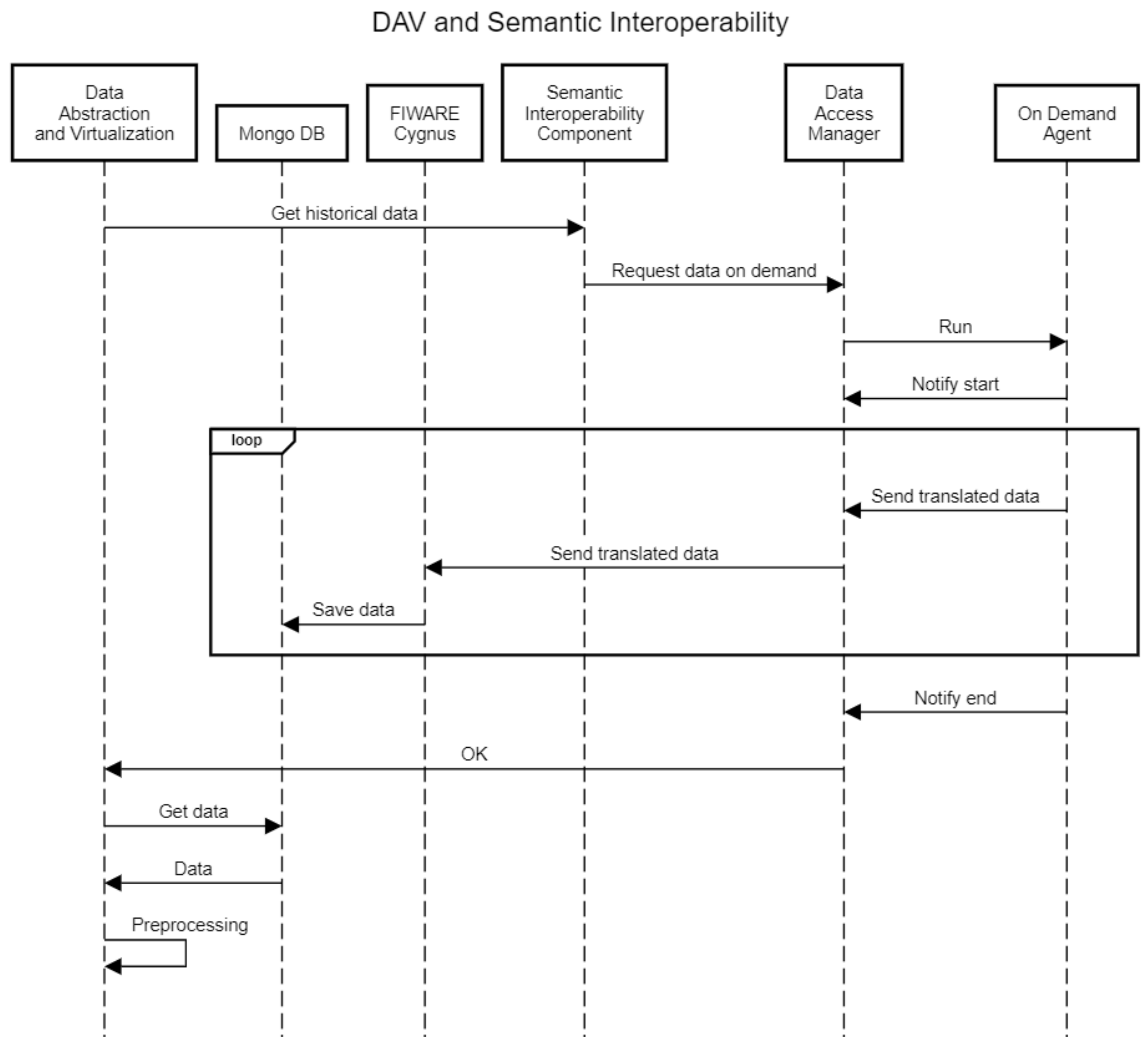


Figure 12 – Integration of the on-demand agents with the DAV component using Cygnus

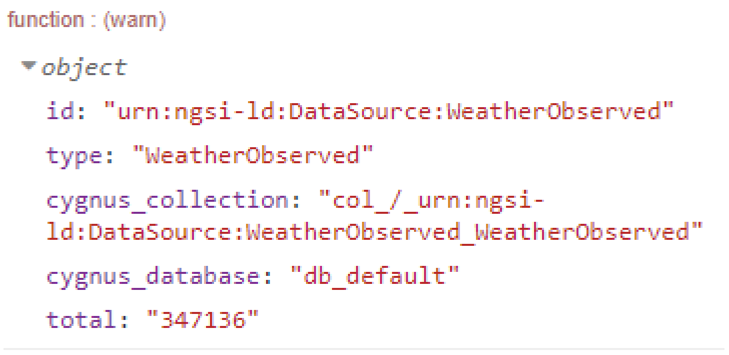
This implementation of Cygnus makes use of MongoDB. All blocks sent to Cygnus are inserted into the same collection if the 'id' field of the data model is not changed. This allows having the data assembled as it was at the beginning.

As this process of sending data to Cygnus can take quite long time depending on the volume of data, an API has been developed that informs the DAV that the data transfer has been completed. The name of this endpoint is datanotify. Next picture depicts an example with the properties that are sent using this endpoint. Properties such as:

- id: Identifier of the data model sent to Cygnus
- type: Type property of the data model
- cygnus_collection: Name of the collection where the entities are stored
- cygnus_database: Name of the database used in MongoDB (by default is always db_default)

[2] http://flume.apache.org/

- total: Total number of records inserted



```
function : (warn)
▾object
  id: "urn:ngsi-ld:DataSource:WeatherObserved"
  type: "WeatherObserved"
  cygnus_collection: "col_/_urn:ngsi-
  ld:DataSource:WeatherObserved_WeatherObserved"
  cygnus_database: "db_default"
  total: "347136"
```

Figure 13 – Example of notification sent to the DAV endpoint

5 SEMANTIC INTEROPERABILITY COMPONENT

This section covers the implementation status of the “Semantic Interoperability Component” in M27 of the project. This component provides a unified API to access the data from the different data sources, as well as a common data model that provides a common understanding of the data; this way, it enables semantic interoperability. The Semantic Interoperability Component has been described from a technical point of view in deliverables D3.2 [16] and D2.4 [6]. Since the present document is an update to D3.2, the focus will be placed on providing an overview of the implementation of the component and the involved technologies. Hence, the following subsections explain the relevance of the Orion Context Broker[3] and the Fiware Ecosystem[4], the software pieces developed, the internal data model definition, the current interaction achieved with other components and the data modelling process defined in the DataPorts project. Section 5.4 covers the expected actions that will be carried out after the release of the platform in M30. Finally, Section 5.5 offers an example scenario to describe in a visual way the functionalities of this component and its interactions with other components of the platform.

5.1 OVERVIEW

The component is a semantic middleware that collects and distributes the data and metadata obtained by the agents deployed in the Data Access Component to the different components of the platform. This layer offers the mechanisms, enablers, and ontologies to achieve semantic interoperability. Specifically, the component offers a framework and API to describe and provide ports’ data together with mappings to standard vocabularies to simplify the reuse of data by other applications and components.

The component mainly performs the following actions:

- Stores locally the metadata of the Data Sources in the Metadata Registry and provides an interface to make the metadata available for all the internal components of the DataPorts Platform.
- Distributes the data to the components of the upper layers. With this aim, it provides:
  - An interface for publishing and subscribing streams of data.
  - An interface for obtaining data on demand. This includes both historical data and last values.
- Validates that the data sent to the data consumers follow the DataPorts Data Model.
- Offers a repository with the DataPorts common Data Model.
- Registers the available data sources in the Data Governance component to provide a description of the data sources for the data consumers from other organisations.
- Interacts with the environment defined in WP4 corresponding with the Data Governance and Security of the DataPorts Platform to enforce the defined access policies and provide a secure access to the data.

Since the “Semantic Interoperability Component” is a middleware component of the platform, its main responsibility is to interact with the rest of the components. This interaction should be done in an efficient and effective way, this means without supposing a bottleneck in the exchange of information between the internal components. This is a fact that must be highlighted and that makes the component a key part of the DataPorts Platform. These relationships have been analysed within the framework of WP2, specifically, during requirements gathering, analysis of expected functionalities, specifications, and architecture design phases. The corresponding WP2 deliverables, like D2.1 [7] and D2.4 [6], provide concrete details about those phases. However, an important part of the work during this period has been the definition of these

[3] https://fiware-orion.readthedocs.io/en/master/

[4] https://www.fiware.org/

relationships from a technical point of view, with the objective of ensuring that the pieces of the platform fit together during the implementation process.

This is mainly illustrated in a joint work with the Data Access component to provide both components fully integrated with each other. The components obtained from these two tasks have major interactions, for example, joint deployments, the use of the same internal data model, common databases, common graphical interface, etc. For this reason, it is difficult to understand the Semantic Interoperability component without the Data Access component and vice versa. Therefore, it is recommended to read both the D3.1 [2] and D3.5 to get the full picture of this joint work.

The technical relationships with the rest of the tasks have been outlined in Section 3. During this period, the joint effort has been focused on ensuring that the API specifications meet the expected requirements, as well as the integration between the Semantic Interoperability component and the other components of the platform. The use of the API will be illustrated in more detail in the descriptions of the components that make use of it, such as the DAV component (described Section 6 of this document) or the Data Analytics services and Cognitive applications (presented in deliverable D3.6 [5]).

Finally, a joint task with the WP2, WP3 and WP5 of the project has been carried out with the aim of offering the common Data Model for the project. Data modelling is a process used to define and analyse data requirements needed to support the business processes within the scope of the corresponding information systems. Therefore, the process of data modelling involves technical partners working closely with business stakeholders, as well as potential users of the information system. The status of this joint task from the technical point of view was reported in the previous version of this deliverable in M18 and is now updated in the present document. The other perspectives of this joint work have been described in D2.2 “Scalability, Interoperability and Definition Standards” (M24) [8] and will be reflected also in the results of the WP5 use cases.

5.2 TECHNOLOGICAL DESCRIPTION

The different blocks that comprise the implementation of the Semantic Interoperability Component are shown in Figure 14. The mappings between the subcomponents of the Semantic Interoperability Component (Figure 3) and the technologies used in their implementation are as follows:

- **Middleware Broker**: Developed using Orion Context Broker and MongoDB. The subcomponents defined for this block are:
    - **Semantic Broker**: The development is focused on an integration with the Data Access Agents (explained in D3.1 [2]) and Orion Context Broker.
    - **Metadata Registry**: The development is focused on an integration with the Data Access Manager (explained in D3.1 [2]), the Data Access Agents (explained in D3.1 [2]) and Orion Context Broker.
    - **Publish/Subscribe**: The subscription mechanism is covered by the functionalities provided by Orion Context Broker. The development effort was focused on the integration with the Data Access Agents (explained in D3.1 [2]).
- **On Demand component**: Is a specific tool developed in Java. The development effort was also focused on an integration with the Data Access Agents (explained in D3.1 [2]) and the Data Access Manager (explained in D3.1 [2]).
- **DataPorts Data Model**: Is hosted in the project Git repository, with a specific organisation and guidelines. The development of the data model has been based on a data modelling methodology defined during the project.
- **Semantic Interoperability API**: Each subcomponent of the Semantic Interoperability Component has its own interface. Their specification is done in Swagger. The interfaces are deployed together using an API Gateway.

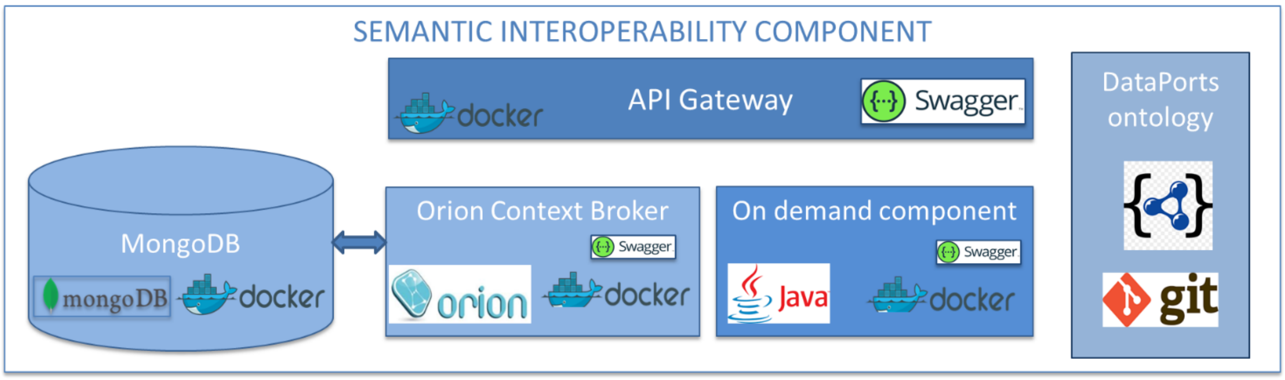


Figure 14 - Semantic Interoperability Component implementation

In addition, other complementary software elements have been developed:

- **Admin User Interface**: Developed in Element and Node.js. This interface has been integrated in the User Interface of the Data Access Manager component (explained in D3.1 [2]).
- **Support tools**: Are a set of pieces of code to support transformations, validations or other specific tasks related with Semantic Interoperability component developed in Java, Python, Node.JS and Angular.

The following subsections provide more specific details of the technology stack.

5.2.1 Orion Context Broker

As a result of the analysis of the requirements and the state-of-the-art of scientific literature, technologies and related EU projects presented in deliverables D2.1 [7], D2.2 [8] and D2.4 [6], the Orion Context Broker[5], which also constitutes the main component in a Powered by Fiware[6] platform, was selected as the core element of the Semantic Interoperability component. The Context Broker is a digital platform component that enables the integration of gathered data including insights for further exploitation. It is important to note that now Orion Context Broker is one of the building blocks of the Connecting Europe Facility (CEF) Digital catalogue[7]. The CEF building blocks provide basic capabilities that can be reused in any European project to ensure interoperability between IT systems and facilitate the implementation of services across borders and sectors. In addition, Fiware NGSI v2 specifications, which are implemented in Orion Context Broker, are compatible with the Context Information Management API standard (NGSI-LD) set by the European Telecommunications Standards Institute (ETSI) [9]. It is important to note that the final version of the common DataPorts Data Model will describe the different entity types both in NGSI v2 and NGSI-LD (Section 5.2.5).

FIWARE Foundation and its Community have been actively contributing to the development of the International Data Spaces Association Reference Architecture Model (IDS-RAM 3.0). Furthermore, the Fiware Foundation is one of the members of the Data Spaces Business Alliance. The alliance aims to drive the adoption of data spaces across Europe and beyond and is composed by Gaia-X, the Big Data Value Association (BDVA), FIWARE Foundation, and the International Data Spaces Association (IDSA) [10]. The Alliance focuses on three major areas:

- Technology and architecture: definition of a common reference model, based on existing architectures and models.

[5] https://fiware-orion.readthedocs.io/en/master/

[6] https://www.fiware.org/

[7] CEF Digital Home (europa.eu)

- Support: provision of resources for the existing organizations and data spaces by pooling their tools, resources and expertise in a focused way.
- Identification and characterization: establishment of a 'Data Spaces Radar' to actively scout potential data spaces.

More specifically, the Orion Context Broker is a key element for the representation of the data sources and concepts relevant to the specific problem tackled. Northbound to the Context Broker, several tools are targeted to support real-time big data processing of the streams of history data generated as digital information evolves over time. The Smart Data Models initiative, launched by the FIWARE Foundation, provides a library of Data Models described in JSON/JSON-LD format which are compatible respectively with the NGSIv2/NGSI-LD APIs to ease the development of portable and interoperable smart solutions in a faster, easier and affordable way, avoiding vendor lock-in scenarios. Data Models published under the initiative are compatible with schema.org and comply with other existing de-facto sectoral standards when they exist. They solve one major issue developers are facing, that is the fact that a given data model specification may be mapped into JSON/JSON-LD in many ways, all of them valid. Smart Data Models initiative facilitates to developers rely on concrete mappings into JSON/JSON-LD, compatible with the NGSIv2/NGSI-LD APIs, that are made available within this library, avoiding interoperability problems derived from alternative mappings [11].

During the first period of the project the Orion Context Broker has been selected as the technological choice to implement the broker. It provides the FIWARE NGSI v2 API which is a Restful API enabling to perform updates, queries or subscribe to changes on context information. However, in the second stage of the project, the Orion-LD Context Broker[8] has been selected as preferable option. This is because it provides an implementation that now is supporting the ETSI NGSI-LD 1.3.1. API specification. It is currently a fork of the original Orion Context Broker extending support to add NGSI-LD and linked data concepts. For that reason, it implements both NGSI-LD and NGSI-v2 APIs.

As Figure 15 shows, Orion manages the data flows from the agents to the components that act as data consumers and provides a NGSI[9] interface. The data managed by Orion is stored in MongoDB[10]. It is important to note that only the last value of each entity is stored. Orion provides a publish/subscribe interface. In addition, the Metadata Registry, which stores locally the metadata of the connected data sources and agents, is managed by Orion. The data provided by Orion follows the DataPorts Data Model (Section 5.2.5).

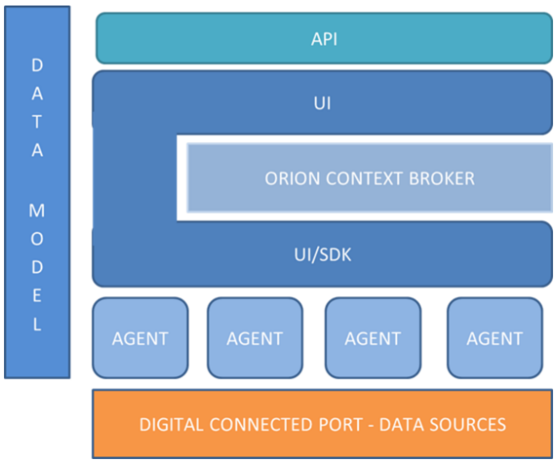


Figure 15 – Relationships between Orion Context Broker and the rest of the components

[8] https://fiware-academy.readthedocs.io/en/latest/core/orion-ld/index.html

[9] https://fiware.github.io/specifications/ngsiv2/stable/

[10] https://www.mongodb.com/

Regarding the Metadata Registry, the metadata is stored in a separate database in MongoDB, named "metadata", which corresponds with the Fiware Service[11] value "metadata". The metadata registry data model includes three data types: "DataSource", "AgentImage" (for the metadata of the on-demand agents) and "AgentContainer" (for the metadata of the publish/subscribe agents); ANNEX A: METADATA REGISTRY provides more details about the metadata. The Semantic Interoperability Component is closely related with the Data Access Component, which is described in detail in deliverable D3.1 [2]. Figure 16 shows a high-level view of the different types of data stored in Orion (including the metadata) as well as the relationships between the data managed by the Semantic Interoperability Component and the data managed by the Data Access Manager, which is part of the Data Access Component. Note that a generic type "Data" has been included in the figure to illustrate the actual data obtained from a data source and represented following the DataPorts Data Model.

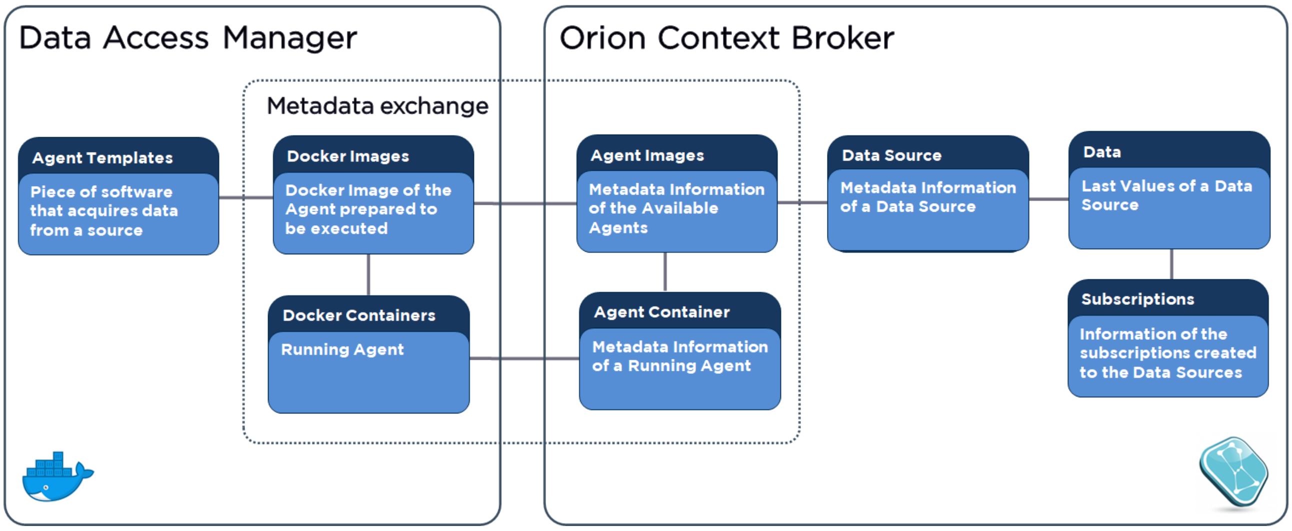

Data Access Manager
Orion Context Broker
Metadata exchange
Agent Templates
Piece of software that acquires data from a source
Docker Images
Docker Image of the Agent prepared to be executed
Docker Containers
Running Agent
Agent Images
Metadata Information of the Available Agents
Agent Container
Metadata Information of a Running Agent
Data Source
Metadata Information of a Data Source
Data
Last Values of a Data Source
Subscriptions
Information of the subscriptions created to the Data Sources

Figure 16 – Relationships between the data managed by Orion and the Data Access Manager

5.2.2 On Demand component

Since the Orion Context Broker only stores the last value of each entity, the On Demand component has been implemented to provide a complementary way to access to the data following an on-demand approach. It includes queries to data sources, like databases, API, or files, that could have historical attributes and offer a huge amount of data related with the same context entity. In addition, this on-demand component provides access to the current contextual values of the entities. This component exposes a REST API, which enables historical data requests, and in addition, it also provides access to the current values using the last values' request. This component is integrated with Orion for accessing the metadata registry and for the last values' requests. Moreover, it communicates with the Data Access component to perform the historical data requests through the on-demand agents. To perform a historical data request, the On Demand component sends a request to the Data Access manager to activate the corresponding on demand agent. Next, the on-demand agent will retrieve the data, translate it and send it to the call-back URL provided in the request, which will typically correspond to an instance of Fiware Cygnus (Section 4.4). It is possible that a single request requires more than one on demand agent to retrieve the data; in that case, the On Demand component will identify the required agents and send the necessary requests to the Data Access Manager. The On Demand component has been implemented in Java and its API has been described in Swagger[12].

[11] https://fiware-orion.readthedocs.io/en/master/user/multitenancy/

[12] https://swagger.io/specification/

5.2.3 Common API

A unified REST API exposes the functions of the Orion Context Broker and the On Demand component. This interface has been implemented using an API gateway, which also provides security by enabling encryption (HTTPS) and performing the communication with the Identity Manager to enable authentication. In addition, it exposes an integrated Swagger description of the Semantic Interoperability Component API. The interfaces covered by the common API are the following:

- Related with the Metadata Registry, the API provides methods for the management of the metadata of the data sources and agents. This includes methods for registering agents and data sources, as well as methods for listing/querying/obtaining the details of an agent or a data source, which include both pull and push mechanisms, and methods for updating or deleting the metadata.
- Related with the On Demand component, the API provides methods for obtaining the current values of the entities, as well as the historical data.
- Related with the Publish/Subscribe Component, the API provides methods for creating subscriptions to the entities, as well as subscription management methods (list subscriptions, retrieve information of a subscription, update/delete subscription).

5.2.4 Deployment of the components

Regarding the deployment of the Semantic Interoperability component, Orion Context Broker, the On Demand component and the Semantic Interoperability API are deployed using Docker[13]. These components can be deployed together using docker-compose[14]. Regarding the Docker images, a precompiled image of the On Demand component is available to ease the deployment of the Semantic Interoperability component. Also, the official Docker images of Orion Context Broker and the API gateway are available on DockerHub[15]. Nevertheless, a Dockerfile is provided with the source code of the On Demand component to allow the creation of custom images (or snapshot images) of the component.

Since there is a close relationship between the Semantic Interoperability component and the Data Access Manager, these two components should be deployed together in order to provide access to the data sources using the Semantic Interoperability API.

5.2.5 Data Modelling process, Data Model Repository and DataPorts Data Model

The DataPorts Data Model is a key element to enable interoperability since it defines a common representation of the information in the DataPorts Platform. The Data Model has been defined considering the application domain, the needs of the pilots and the existing ontologies and data models related with the domains that must be covered by the DataPorts Data Model. The methodology that has been followed to define the Data Model is based on principles provided from other initiatives [12] [13] and comprises the following steps:

1. Identifying the key concepts needed in the Data Model from the scenarios defined in the pilots (which were introduced in deliverable D5.1 [14]).
2. Classifying those concepts as entities, attributes, and relationships.
3. Creating UML diagrams representing the resulting entities and relationships for each of the scenarios of the pilots.
4. Combining the results of the analysis of the scenarios to obtain a high-level view of the Data Model.
5. Classifying the entities of the common Data Model into a set of domains and subjects.

[13] https://www.docker.com/

[14] https://docs.docker.com/compose/

[15] https://hub.docker.com/

6. Analysing the existing ontologies describing the identified domains and subjects to find out which definitions could be reused in the DataPorts Data Model.
7. Defining the detailed specifications of the Data Model.

The DataPorts Data Model integrates concepts from existing ontologies and data models, including the Fiware Smart Data Models[16], the UN/CEFACT[17] data model, the SAREF[18] ontology and the IDSA Information Model[19]. The common Data Model is hosted in a Git repository, which contains the corresponding JSON schema documents describing the syntax in NGSI, as well as the documentation and examples. For each class in the Data Model, normalised (NGSI v2) and simplified (JSON) examples are provided. The Data Model documentation will include equivalence tables to describe the mappings between the DataPorts Data Model and the existing ontologies integrated in the Data Model. In addition, the documentation will include the description of the Data Model using UML diagrams. According to the roadmap, the final version of the Data Model will be based on NGSI-LD[20], following the context information management standard defined by the ETSI (Section 5.2.1). Thus, once NGSI-LD has been adopted, the Data Model repository will include JSON-LD context documents, as well as NGSI-LD and JSON-LD examples, in addition to the current descriptions and examples. It is important to note that the NGSI-LD context facilitates the reuse of the existing Smart Data Models to create data models for new scenarios in the following ways:

- The existing concepts can be extended with new attributes or metadata.
- New concepts can be defined as subclasses of existing concepts.
- Since some attributes of the existing concepts are optional, the attributes that are not valid for a particular scenario can be removed. Similarly, the enumerated values that are redundant in a particular scenario can be removed from the NGSI-LD context.

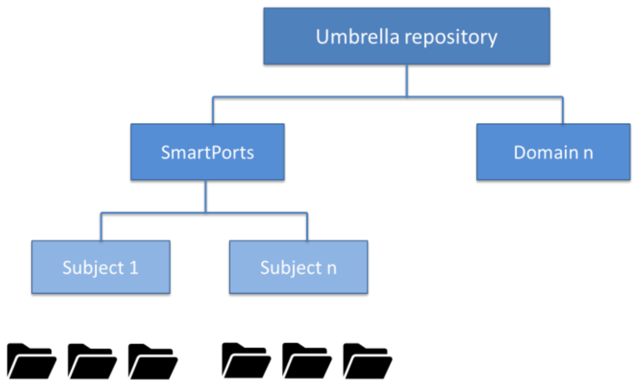


Figure 17 – Initial idea for the structure of the Data Model Git repository

The concepts defined in the Data Model have been organised by domain and subject, and this hierarchy is reflected in the organisation of the Data Model repository. This structure has been designed with the objective of facilitating the future integration of the DataPorts Data Model with the Smart Data Models initiative[21], which is a Fiware-related initiative for the publication of the models, describing data that participants of data spaces can exchange. Thus, the Git repository[22] has a tree structure similar to the one of

[16] https://www.fiware.org/developers/data-models/

[17] https://umm-dev.org/about-umm/

[18] https://saref.etsi.org/

[19] https://github.com/International-Data-Spaces-Association/InformationModel

[20] https://www.etsi.org/committee/cim?jjj=1620912114757

[21] https://smartdatamodels.org/

[22] https://egitlab.iti.es/dataports/data_processing/datamodel

the Smart Data Models Git repositories (Figure 17), in which the root (Umbrella repository) provides access to the repositories of the different domains and contains general information and guidelines, as well as common definitions and resources. As a result of the previous analysis, only one domain (Smart Ports) has been identified. This domain provides access to a set of subjects (Figure 18) and contains shared information and resources of the domain. Similarly, each subject contains folders for each of its entity types (Figure 19), as well as shared information and resources of the subject, such as the NGSI-LD context document, where the rules to interpret the data according to the ontology are described. Finally, the specifications, examples, and information of each entity type are provided in its corresponding folder. Finally, each entity type defined in the Data Model will go through two states: incubated (that is, with existing specifications but not officially adopted by the pilots) and adopted (consolidated entity definitions being actively used in DataPorts).

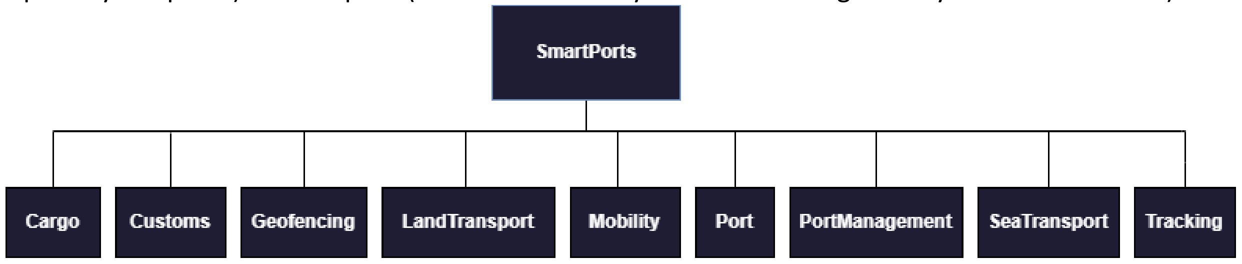


Figure 18 – Domain and subjects of the common Data Model

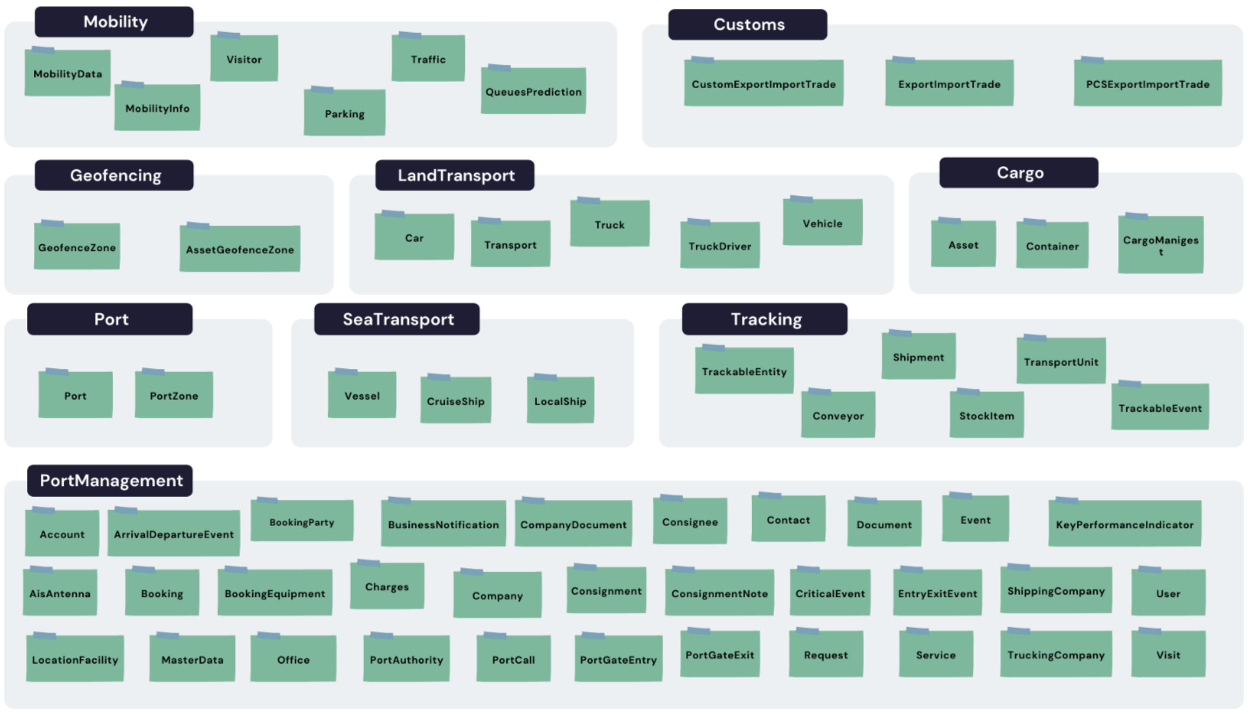
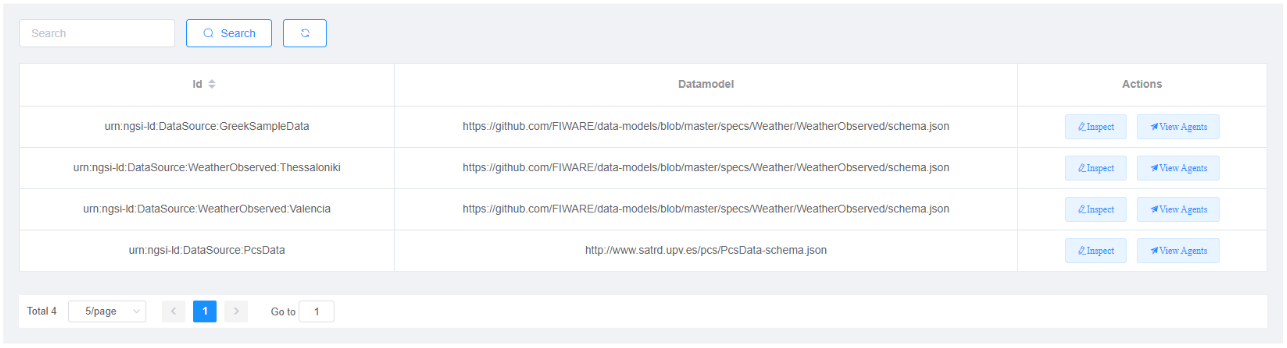


Figure 19 – Subjects and entity types of the common Data Model

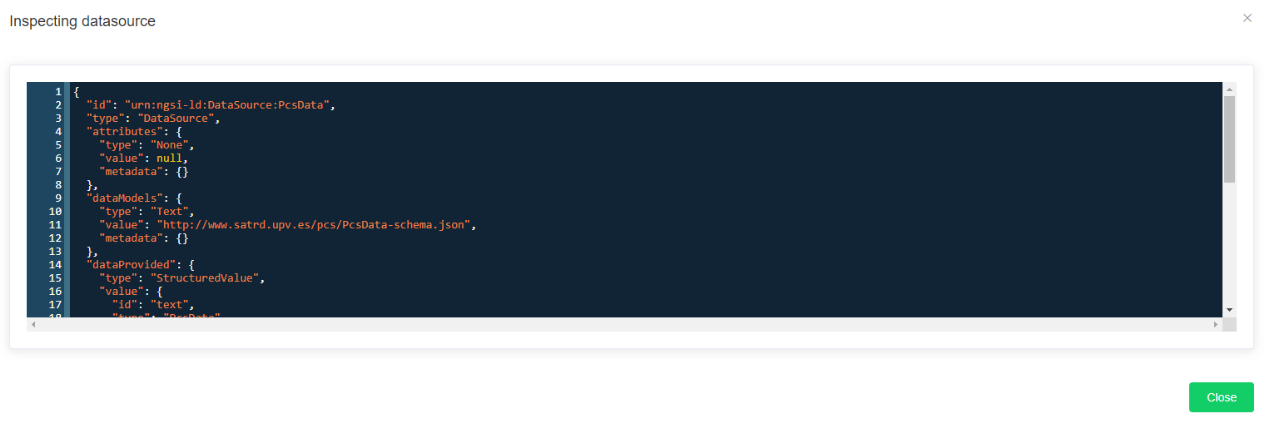
5.2.6 Administration User Interface

To facilitate the management of the data sources and the subscriptions, an administration User Interface has been developed. This interface provides a general view of the connected data sources (Figure 20), as well as the metadata of the selected data source (Figure 21) and information about which registered agents can be used for accessing the data source. Similarly, the Admin UI provides a general view of the existing subscriptions (Figure 22), as well as the detailed information of the selected subscription (Figure 23). The Administration UI has been integrated with the Data Access Manager web UI (described in D3.1 [2]) in a new tab called "Orion". Section 5.5 contains some examples of the use of this interface.



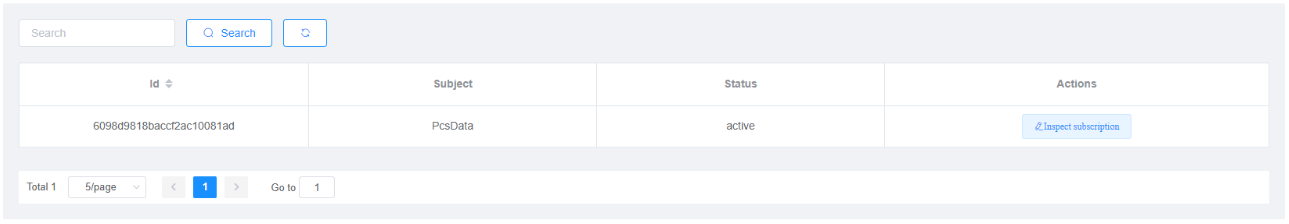
Id	Datamodel	Actions
urn:ngsi-ld:DataSource:GreekSampleData	https://github.com/FIWARE/data-models/blob/master/specs/Weather/WeatherObserved/schema.json	Inspect View Agents
urn:ngsi-ld:DataSource:WeatherObserved:Thessaloniki	https://github.com/FIWARE/data-models/blob/master/specs/Weather/WeatherObserved/schema.json	Inspect View Agents
urn:ngsi-ld:DataSource:WeatherObserved:Valencia	https://github.com/FIWARE/data-models/blob/master/specs/Weather/WeatherObserved/schema.json	Inspect View Agents
urn:ngsi-ld:DataSource:PcsData	http://www.satrd.upv.es/pcs/PcsData-schema.json	Inspect View Agents

Figure 20 – Data sources metadata shown in the Admin UI



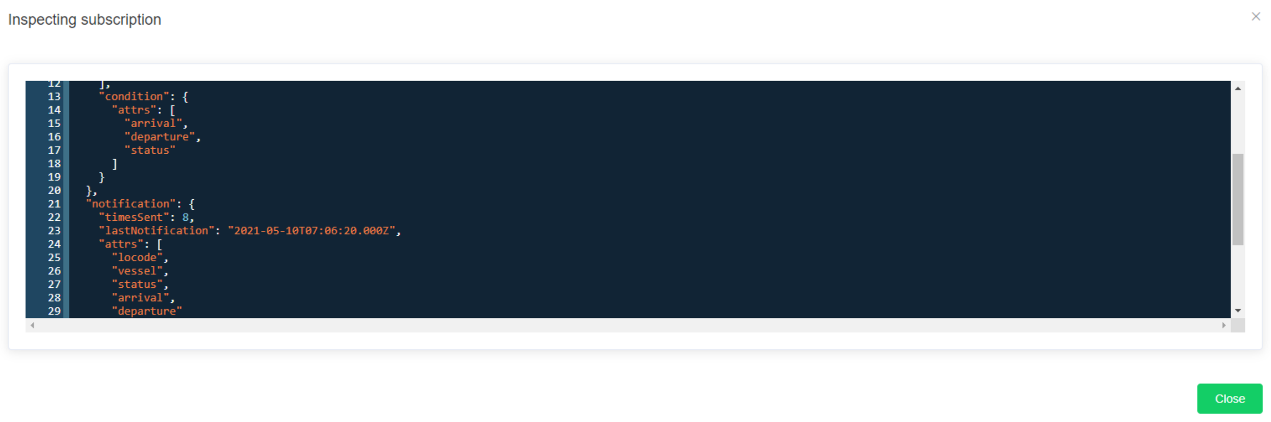
```
{
  "id": "urn:ngsi-ld:DataSource:PcsData",
  "type": "DataSource",
  "attributes": {
    "type": "None",
    "value": null,
    "metadata": {}
  },
  "dataModels": {
    "type": "Text",
    "value": "http://www.satrd.upv.es/pcs/PcsData-schema.json",
    "metadata": {}
  },
  "dataProvided": {
    "type": "StructuredValue",
    "value": {
      "id": "text",
      "type": "PcsData",
```

Figure 21 – Admin UI modal box of an example data source



Id	Subject	Status	Actions
6098d9818baccf2ac10081ad	PcsData	active	Inspect subscription

Figure 22 – Subscriptions in Orion Context Broker (shown in the Admin UI)



```
      ],
      "condition": {
        "attrs": [
          "arrival",
          "departure",
          "status"
        ]
      }
    },
    "notification": {
      "timesSent": 8,
      "lastNotification": "2021-05-10T07:06:20.000Z",
      "attrs": [
        "locode",
        "vessel",
        "status",
        "arrival",
        "departure"
```

Figure 23 – Admin UI modal box of a subscription

5.2.7 Support tools

In addition to the components described in the previous subsections, a set of support tools are provided to facilitate the learning and adoption of the Semantic Interoperability component. This category includes the following tools:

- Graphical tool for testing, learning and validation: this tool has been developed to allow an easy way to visualise the data. This tool is different from the Administration UI presented in Section 5.2.6, which is focused on the metadata, while the objective of the tool presented in this section is to provide a graphical tool that makes use of the API operations presented in Section 5.2.3 to access the data. Hence, this tool offers access to the historical data and last values, as well as access to near real-time data using subscriptions. In addition, the tool offers widgets to create simple visualisations of the data; however, it would be necessary to modify the source code of the application to adapt the visualisation to the specific type of data. This tool can be used for testing, demonstration or learning purposes. The demonstration presented in Section 5.5 makes use of this tool to illustrate several scenarios.
- Data Modelling and Validation tools:
  - Data model definition template: is an Excel file that can be converted to a JSON schema. It allows the definition of new entities of the Data Model in an easy way.
  - JSON schema generator: is a Python script that generates the JSON schema of an entity type from the input provided in the Excel template.
  - JSON schema validation tool: checks if the provided JSON schema is valid.
  - Data format validation tool (based on JSON schema): checks if the data follow the definition of the entity type provided in the Data Model.
  - JSON to JSON schema converter: generates the JSON schema that describes the input data. This tool allows the definition of the JSON schema of an entity type using an example as input.
- Node-RED nodes[23] for the main functions of the Semantic Interoperability API, which will support the authentication with the Identity Manager of the DataPorts platform.

5.3 DEVELOPMENT STATUS (M27)

As indicated in previous sections, the development of the components follows the requirements and functionalities described in D2.1 [7] and D2.4 [6], respectively. The first functional version of the component was delivered in M18. The work during that first period was mainly centred on the interaction with T3.1 because this task ended in the M18 of the project. The objective was to offer a joint deployment that guarantees a functional Data Access Layer through an interoperable API. After M18, the development effort was focused on the data model, as well as the integration with the Data Governance and the upper layers of the platform.

As a result of these efforts, the Semantic Interoperability Component has been developed. The component is integrated with the Data Access Component, as well as the upper layers of the platform, and ready to be used by the WP5 Use Cases and the applications built on top of the platform. The implementation of the component includes the following features:

- Framework to describe and provide ports data together with mappings to standard vocabularies.
- Orion Context Broker as core component, using NGSI v2. The final version of the component will make use of Orion-LD as core component, which is an evolution of Orion Context Broker that supports NGSI-LD in addition to NGSI v2. The motivation is that ETSI NGSI-LD is implemented as a standard mechanism for effective data exchange between different stakeholders.

[23] https://flows.nodered.org/

- Common interoperable API available providing the interfaces listed in Section 5.2.3.
- Metadata Registry, implemented according to the internal data model (ANNEX A: METADATA REGISTRY).
- Functional integration with the Data Access Manager following the defined internal data model.
- Functional integration with the Data Access Agents. It guarantees the acquisition and distribution of data and metadata from the heterogeneous data sources using the Semantic Interoperability API.
- Publish/subscribe interface, provided by the Orion Context Broker.
- Access to data on demand, enabled by the On Demand component. This includes access to historical data and current values.
- Unified packaging and deployment on Docker together with the Data Access Component. The components can be deployed in a PC, Server, Cloud Infrastructure or in a Raspberry Pi.
- Admin User Interface to visualise the information of the metadata registry and the subscriptions available. This interface has been integrated with the Data Access Manager User Interface.
- A set of support tools developed to facilitate the use and learning of the component.

In addition, the following development and integration tasks have been carried out:

- Definition and development of the internal Data Model of the component. Provision of data modelling support tools to facilitate the implementation of the common data model.
- Definition of the common DataPorts data model following a data modelling methodology defined in the previous period, as well as the guidelines for the specifications of the entity types and the repositories. The definition of the common Data Model has been achieved through a joint work with WP2, WP3 and WP5 partners. Defining NGSI-LD specifications of the DataPorts Data Model. Provision of support tools for the adaptation of the common data model to NGSI-LD.
- Integration of the Semantic Component API in the DataPorts Platform common API.
- Initial integration of the Semantic Interoperability component with the security and data governance blocks of the DataPorts platform.
- Integration between the Metadata Registry and the Data Governance component.
- Implementation of a graphical testing tool to validate, demonstrate, and teach about the functionalities of the component.
- Integration between the components implemented in Tasks T3.2 (Semantic Interoperability Component) and T3.3 (Data Abstraction and Virtualization component).
- Integration of the component with external added value services like Node-RED[24], Apache NiFi[25], CKAN[26], Wirecloud[27], etc.

Regarding the plan versions of the Semantic Interoperability Component until M30 (final release of the platform), the roadmap consists of the following expected tasks:

- Complete integration of the Semantic Interoperability Component with Privacy, Security and Blockchain. This task will be carried out in collaboration with WP4.
- Improving the Semantic Interoperability component implementation during the Use Cases deployment period considering the needs that may arise during that period. Improving the ease of use. Offering several releases of the component.
- The adoption of the NGSI-LD meta-model formally defines the following NGSI-LD foundational concepts (Entities, Relationships, Properties) based on RDF/RDFS/OWL and JSON-LD in the DataPorts

[24] https://nodered.org/

[25] https://nifi.apache.org/

[26] https://fiware-ckan-extensions.readthedocs.io/en/latest/

[27] https://wirecloud.readthedocs.io/en/stable/

Data Model. It guarantees to take advantage of NGSI-LD as a mechanism to enhance context data entities through adding the concept of linked data. The work during the following month will be focused on improving the provision of REST API interaction with Linked Data through the component, including the semantic functionalities identified or required.
- Focus on the scalability, extensibility, and modularity of the component.

5.4 FUTURE WORK: ROADMAP

The Semantic Interoperability component brings the essential data models, mechanisms and enablers to provide interoperability with the current and future data sources of the seaport's environment. In addition, it has the specific objective of offering a solution compatible with the solutions that the Fiware technical stack will offer in the coming months in various areas such as digital twins, data spaces or Blockchain.

The open-source nature of the involved technologies leads to a strong interest in maintaining and upgrading the component constantly. Therefore, the future work seeks to offer the novelties developed to the potential developers of these open-source communities, as well as update the developed solution to be compatible with future enablers and tools. For that reason, the future roadmap of the component is focused on the following main aspects:

- Open-source release of the developed novel components like On Demand Component and support tools.
- Contributing to the Smart Data Models initiative with the concepts defined in the common DataPorts data model. The aim is to participate in the periodical meetings and to collaborate actively in the subject related to Ports.
- Participation in the Data Spaces Business Alliance future activities, which is an initiative promoted in the scope of the Fiware, Data Spaces, Gaia-X, and BDVA activities. This initiative promotes the collaboration between organisations and projects sharing their vision on how to materialise an open standard-based, open source available and CEF-compatible software infrastructure for creation of data spaces in Europe.

5.5 EXAMPLE OF USE: DEMONSTRATION

The following demonstration scenarios highlight the Semantic Interoperability component main functionalities, already described in the previous subsections. The sections from Section 5.5.1 to Section 5.5.4 are focused on illustrating the mechanisms, enablers, data models, ontologies and interfaces offered to facilitate the Common Access and Management of the different Data Sources integrated into the DataPorts Platform. Section 5.5.5 and Section 5.5.6 cover the main functionalities offered to guarantee the integration of the component in the DataPorts technological stack. Finally, Section 5.5.7 highlights the functionalities offered from the point of view of a deployment (together with the data access component) as an autonomous component focused on offering added value to other end user applications.

5.5.1 Demonstration overview

Figure 24 shows an overview of the sample scenarios provided by the demonstration to explain the functionalities related with the Semantic Interoperability component. It mainly covers the flow of data and metadata between the Data Access and Data Semantic Interoperability services to other internal components or services and applications. More specifically, several data sources with simulated data are connected to the Semantic Interoperability Component through the corresponding agents, which are managed by the Data Access Component. This way, the data can be accessed through the Semantic Interoperability Component API. In addition, a demo application (presented in Section 5.2.7) has been provided to show in a clear and visual manner the different ways the data can be obtained.

The potential users of the platform for whom this demonstration is intended are the following:

- Data providers and Data Owners: The demonstration shows how the Semantic Interoperability Component provides access to the data and metadata provided by the agents.
- System Integrators: The demonstration offers to the Agent Developers a clear view of the interoperability functionalities offered by the platform once an agent is deployed and running.
- DataPorts Platform administrators: The demonstration shows to the platform administrators a clear view of how to manage and visualise the agents and the data sources.
- DataPorts Platform developers: The demonstration explains how the internal components access to the Data Sources available through the Semantic Interoperability Component and the functionalities of the API.
- Data Users: The demonstration offers insights of the advantages of having a common API and Data Model to make use of the data through new or existing applications.

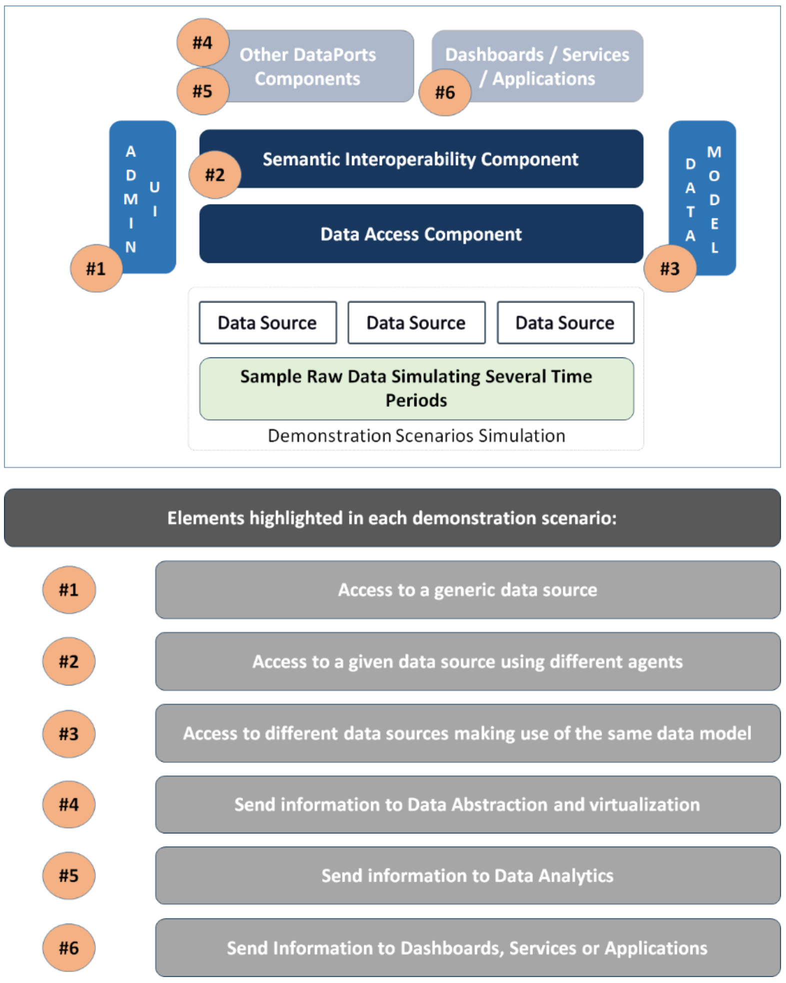

#4
Other DataPorts Components
#5
Dashboards / Services / Applications
#6
ADMIN UI
#1
Semantic Interoperability Component
#2
Data Access Component
DATA MODEL
#3
Data Source
Data Source
Data Source
Sample Raw Data Simulating Several Time Periods
Demonstration Scenarios Simulation
Elements highlighted in each demonstration scenario:
#1 Access to a generic data source
#2 Access to a given data source using different agents
#3 Access to different data sources making use of the same data model
#4 Send information to Data Abstraction and virtualization
#5 Send information to Data Analytics
#6 Send Information to Dashboards, Services or Applications

Figure 24 – Demonstration overview

The demonstration includes the following features:

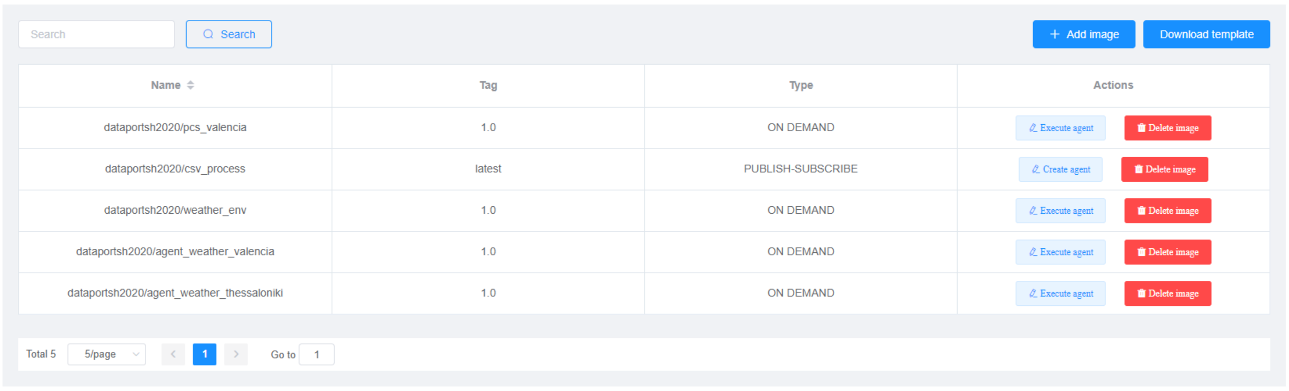
- Local metadata storage/management and Data Models.
- Access to real-time and historic data.
- Subscription to a data source.
- Internal components integration.
- Use of the data by external applications.
- Reuse of data by existing tools.

The sample data sources and agents used to cover the different scenarios of the demonstration are:

- A generic data source, connected through a generic agent and sending dummy data.
- Port Community System (PCS) data. PCS provides data about port and logistic processes. In the example, port calls were used as input data. The demonstration includes two different agents for the PCS data: a publish/subscribe agent, which enables access to the current values (providing the necessary updates), and an on-demand agent for historical data.
- Weather data sources, connected through their corresponding on demand agents. These agents make use of the WeatherObserved data model[28] from Fiware. In the example, three data sources were used, which provided data from Valencia, Thessaloniki, and several locations in Greece, respectively.

5.5.2 Scenario #1: access to a generic data source

The first scenario shows the available data sources, once the corresponding agents have been created (following the steps described in deliverable D3.1 [2]) and uploaded to the platform. The different data sources (Figure 20) and agents (Figure 25), which were introduced in the previous Section, appear in the corresponding screens of the Admin UI ready to be accessed. Next scenarios are going to provide more details about how the data of these data sources are obtained.

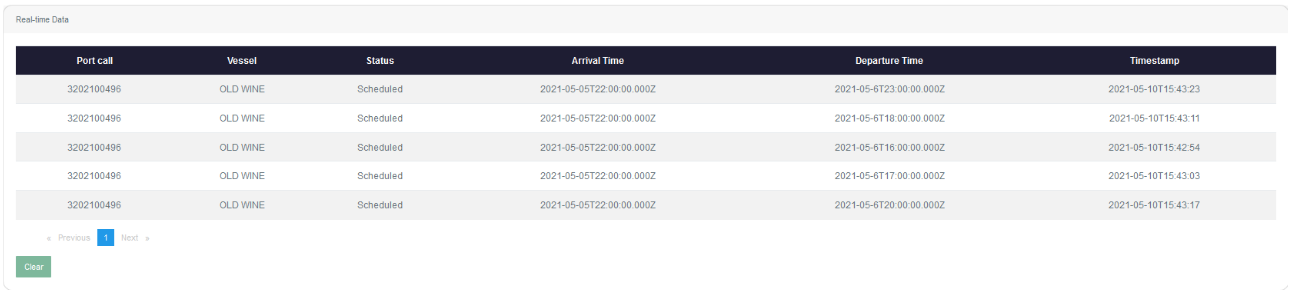


Name	Tag	Type	Actions
dataportsh2020/pcs_valencia	1.0	ON DEMAND	Execute agent Delete image
dataportsh2020/csv_process	latest	PUBLISH-SUBSCRIBE	Create agent Delete image
dataportsh2020/weather_env	1.0	ON DEMAND	Execute agent Delete image
dataportsh2020/agent_weather_valencia	1.0	ON DEMAND	Execute agent Delete image
dataportsh2020/agent_weather_thessaloniki	1.0	ON DEMAND	Execute agent Delete image

Figure 25 – Available agents (shown in the Data Access Manager UI)

5.5.3 Scenario #2: access to a data source using different types of agents

This scenario shows the different ways to obtain data from a given data source. The selected data source for this scenario provides a sample of PCS data and is connected to the DataPorts Platform using two different agents. The following functionalities of the Semantic Interoperability component are demonstrated.



Port call	Vessel	Status	Arrival Time	Departure Time	Timestamp
3202100496	OLD WINE	Scheduled	2021-05-05T22:00:00.000Z	2021-05-6T23:00:00.000Z	2021-05-10T15:43:23
3202100496	OLD WINE	Scheduled	2021-05-05T22:00:00.000Z	2021-05-6T18:00:00.000Z	2021-05-10T15:43:11
3202100496	OLD WINE	Scheduled	2021-05-05T22:00:00.000Z	2021-05-6T16:00:00.000Z	2021-05-10T15:42:54
3202100496	OLD WINE	Scheduled	2021-05-05T22:00:00.000Z	2021-05-6T17:00:00.000Z	2021-05-10T15:43:03
3202100496	OLD WINE	Scheduled	2021-05-05T22:00:00.000Z	2021-05-6T20:00:00.000Z	2021-05-10T15:43:17

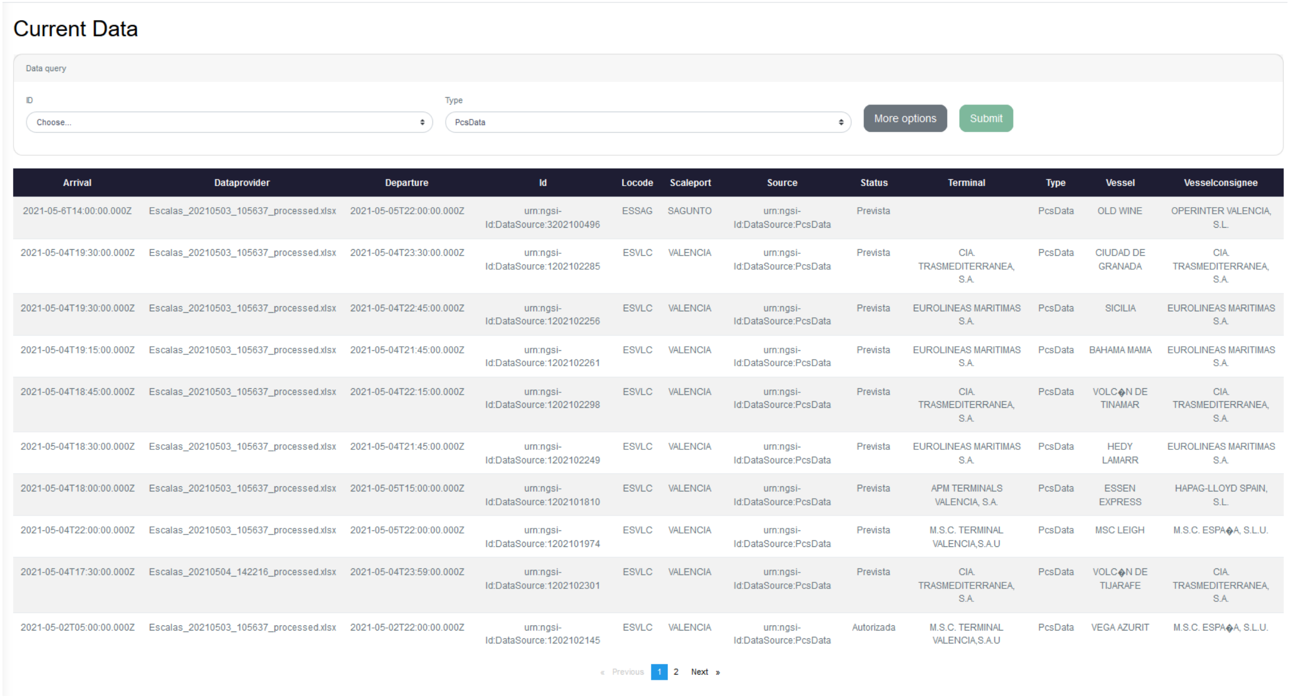
Figure 26 – Notifications (port call status changes) received in the demo application

Firstly, publish/subscribe access is demonstrated. The demo application subscribes to the PCS data and Orion Context Broker sends notifications when there is a change in any of the parameters of interest; in this example, the parameter of interest is the status of the port calls. The received notifications are shown in the

[28] https://github.com/smart-data-models/dataModel.Weather/tree/master/WeatherObserved

demo application UI (Figure 26) and the details of the existing subscriptions can be shown in the Data Access Manager UI (Figure 22 and Figure 23).

Next, the access to the last values of the data is shown. In this example, the latest received PCS data, which is stored in Orion and shows the status of the port, is requested to the Semantic Interoperability component. The demo application sends a last value request to the On Demand component and shows the result (Figure 27), which in this example represents the status of the port calls.

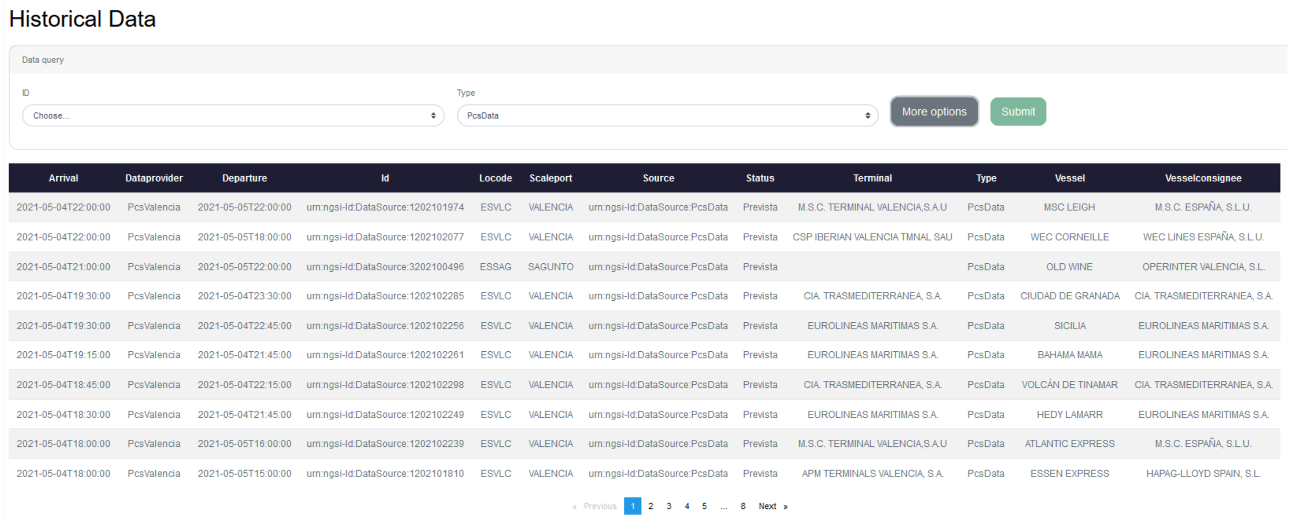


Arrival	Dataprovider	Departure	Id	Locode	Scaleport	Source	Status	Terminal	Type	Vessel	Vesselconsignee
2021-05-6T14:00:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-05T22:00:00.000Z	urn:ngsi-Id:DataSource:3202100496	ESSAG	SAGUNTO	urn:ngsi-Id:DataSource:PcsData	Prevista		PcsData	OLD WINE	OPERINTER VALENCIA, S.L.
2021-05-04T19:30:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-04T23:30:00.000Z	urn:ngsi-Id:DataSource:1202102285	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	CIA. TRASMEDITERRANEA, S.A.	PcsData	CIUDAD DE GRANADA	CIA. TRASMEDITERRANEA, S.A.
2021-05-04T19:30:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-04T22:45:00.000Z	urn:ngsi-Id:DataSource:1202102256	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	EUROLINEAS MARITIMAS S.A.	PcsData	SICILIA	EUROLINEAS MARITIMAS S.A.
2021-05-04T19:15:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-04T21:45:00.000Z	urn:ngsi-Id:DataSource:1202102261	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	EUROLINEAS MARITIMAS S.A.	PcsData	BAHAMA MAMA	EUROLINEAS MARITIMAS S.A.
2021-05-04T18:45:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-04T22:15:00.000Z	urn:ngsi-Id:DataSource:1202102298	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	CIA. TRASMEDITERRANEA, S.A.	PcsData	VOLC�N DE TINAMAR	CIA. TRASMEDITERRANEA, S.A.
2021-05-04T18:30:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-04T21:45:00.000Z	urn:ngsi-Id:DataSource:1202102249	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	EUROLINEAS MARITIMAS S.A.	PcsData	HEDY LAMARR	EUROLINEAS MARITIMAS S.A.
2021-05-04T18:00:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-05T15:00:00.000Z	urn:ngsi-Id:DataSource:1202101810	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	APM TERMINALS VALENCIA, S.A.	PcsData	ESSEN EXPRESS	HAPAG-LLOYD SPAIN, S.L.
2021-05-04T22:00:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-05T22:00:00.000Z	urn:ngsi-Id:DataSource:1202101974	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	M.S.C. TERMINAL VALENCIA,S.A.U	PcsData	MSC LEIGH	M.S.C. ESPA�A, S.L.U.
2021-05-04T17:30:00.000Z	Escalas_20210504_142216_processed.xlsx	2021-05-04T23:59:00.000Z	urn:ngsi-Id:DataSource:1202102301	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	CIA. TRASMEDITERRANEA, S.A.	PcsData	VOLC�N DE TIJARAFE	CIA. TRASMEDITERRANEA, S.A.
2021-05-02T05:00:00.000Z	Escalas_20210503_105637_processed.xlsx	2021-05-02T22:00:00.000Z	urn:ngsi-Id:DataSource:1202102145	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Autorizada	M.S.C. TERMINAL VALENCIA,S.A.U	PcsData	VEGA AZURIT	M.S.C. ESPA�A, S.L.U.

« Previous 1 2 Next »

Figure 27 – Current PCS data (port calls) shown in the demo application

Finally, the access to historical PCS data is demonstrated. The demo application sends a historical data request to the On Demand component and shows the received data (Figure 28).

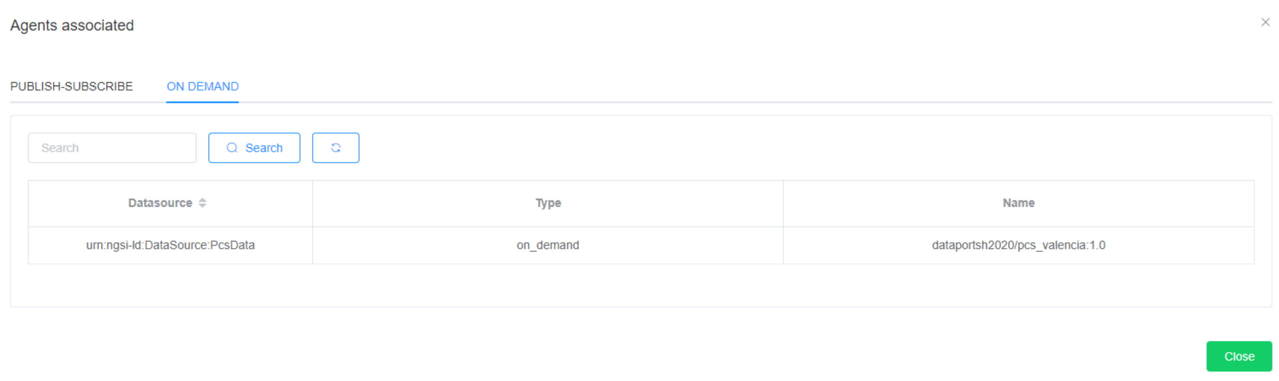


Arrival	Dataprovider	Departure	Id	Locode	Scaleport	Source	Status	Terminal	Type	Vessel	Vesselconsignee
2021-05-04T22:00:00	PcsValencia	2021-05-05T22:00:00	urn:ngsi-Id:DataSource:1202101974	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	M.S.C. TERMINAL VALENCIA,S.A.U	PcsData	MSC LEIGH	M.S.C. ESPAÑA, S.L.U.
2021-05-04T22:00:00	PcsValencia	2021-05-05T18:00:00	urn:ngsi-Id:DataSource:1202102077	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	CSP IBERIAN VALENCIA TMNAL SAU	PcsData	WEC CORNEILLE	WEC LINES ESPAÑA, S.L.U.
2021-05-04T21:00:00	PcsValencia	2021-05-05T22:00:00	urn:ngsi-Id:DataSource:3202100496	ESSAG	SAGUNTO	urn:ngsi-Id:DataSource:PcsData	Prevista		PcsData	OLD WINE	OPERINTER VALENCIA, S.L.
2021-05-04T19:30:00	PcsValencia	2021-05-04T23:30:00	urn:ngsi-Id:DataSource:1202102285	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	CIA. TRASMEDITERRANEA, S.A.	PcsData	CIUDAD DE GRANADA	CIA. TRASMEDITERRANEA, S.A.
2021-05-04T19:30:00	PcsValencia	2021-05-04T22:45:00	urn:ngsi-Id:DataSource:1202102256	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	EUROLINEAS MARITIMAS S.A.	PcsData	SICILIA	EUROLINEAS MARITIMAS S.A.
2021-05-04T19:15:00	PcsValencia	2021-05-04T21:45:00	urn:ngsi-Id:DataSource:1202102261	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	EUROLINEAS MARITIMAS S.A.	PcsData	BAHAMA MAMA	EUROLINEAS MARITIMAS S.A.
2021-05-04T18:45:00	PcsValencia	2021-05-04T22:15:00	urn:ngsi-Id:DataSource:1202102298	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	CIA. TRASMEDITERRANEA, S.A.	PcsData	VOLCÁN DE TINAMAR	CIA. TRASMEDITERRANEA, S.A.
2021-05-04T18:30:00	PcsValencia	2021-05-04T21:45:00	urn:ngsi-Id:DataSource:1202102249	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	EUROLINEAS MARITIMAS S.A.	PcsData	HEDY LAMARR	EUROLINEAS MARITIMAS S.A.
2021-05-04T18:00:00	PcsValencia	2021-05-05T16:00:00	urn:ngsi-Id:DataSource:1202102239	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	M.S.C. TERMINAL VALENCIA,S.A.U	PcsData	ATLANTIC EXPRESS	M.S.C. ESPAÑA, S.L.U.
2021-05-04T18:00:00	PcsValencia	2021-05-05T15:00:00	urn:ngsi-Id:DataSource:1202101810	ESVLC	VALENCIA	urn:ngsi-Id:DataSource:PcsData	Prevista	APM TERMINALS VALENCIA, S.A.	PcsData	ESSEN EXPRESS	HAPAG-LLOYD SPAIN, S.L.

« Previous 1 2 3 4 5 ... 8 Next »

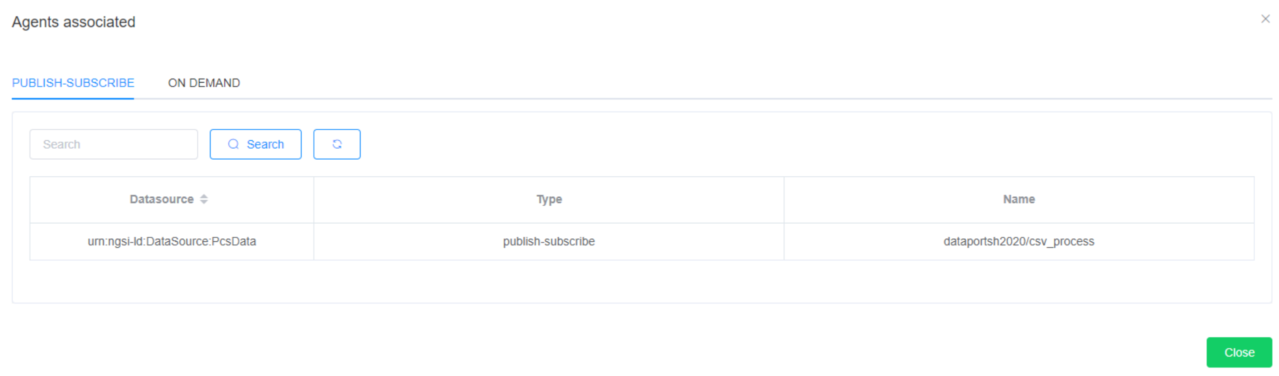
Figure 28 – Historical PCS data (port calls) shown in the demo application

In addition, the metadata of the data source (Figure 21) and agents (Figure 29 and Figure 30) can be shown in the Data Access Manager UI.



Datasource	Type	Name
urn:ngsi-ld:DataSource:PcsData	on_demand	dataportsh2020/pcs_valencia:1.0

Figure 29 – On-demand agent registered for the example data source

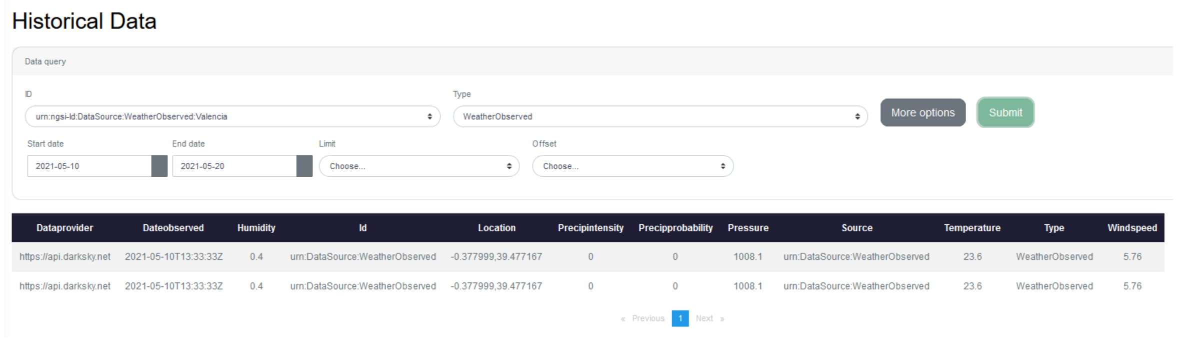


Datasource	Type	Name
urn:ngsi-ld:DataSource:PcsData	publish-subscribe	dataportsh2020/csv_process

Figure 30 – Publish/subscribe agent registered for the example data source

5.5.4 Scenario #3: Access to different data sources that are using the same Data Model

In this example, three different data sources, which provide weather data from Valencia, Thessaloniki, and several locations in Greece, respectively, are connected to the DataPorts Platform through their corresponding on-demand agents. These agents make use of the WeatherObserved data model[29] from Fiware to represent the data. Each agent has defined its corresponding methods to access to the raw data, which depend on the type of data source (more details in deliverable D3.1 [2]). For example, in the sample scenario, two of them make use of an API (which returns data in a non-standardised JSON format), while another obtains the data from an Excel file (which contains the data in a non-standardised CSV format). This process is transparent to the users, they only need to retrieve the data through the Semantic Interoperability component. Then, the users can know the concrete specification of the data. Figure 31, Figure 32 and Figure 33 show examples of the historical data obtained from the example data sources in this scenario.

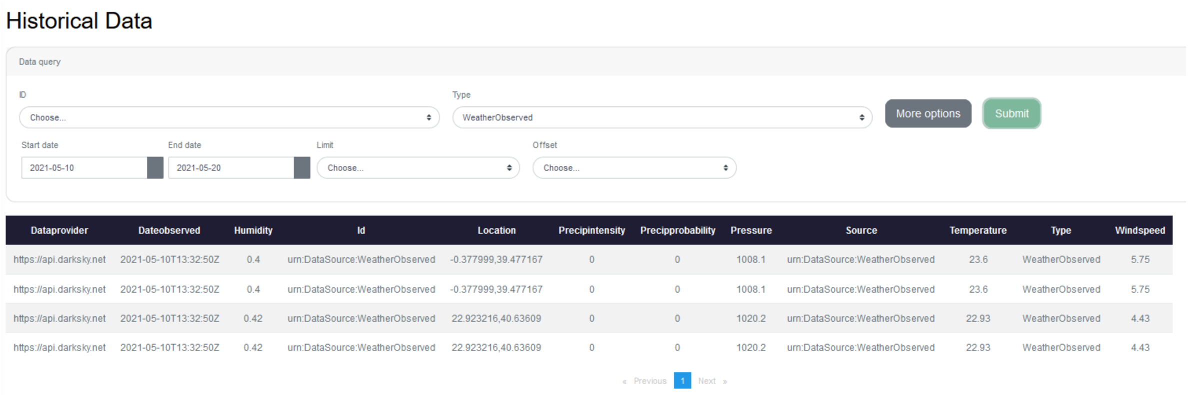


ID: urn:ngsi-ld:DataSource:WeatherObserved:Valencia
Type: WeatherObserved
Start date: 2021-05-10
End date: 2021-05-20
Limit: Choose...
Offset: Choose...

Dataprovider	Dateobserved	Humidity	Id	Location	Precipintensity	Precipprobability	Pressure	Source	Temperature	Type	Windspeed
https://api.darksky.net	2021-05-10T13:33:33Z	0.4	urn:DataSource:WeatherObserved	-0.377999,39.477167	0	0	1008.1	urn:DataSource:WeatherObserved	23.6	WeatherObserved	5.76
https://api.darksky.net	2021-05-10T13:33:33Z	0.4	urn:DataSource:WeatherObserved	-0.377999,39.477167	0	0	1008.1	urn:DataSource:WeatherObserved	23.6	WeatherObserved	5.76

Figure 31 – Historical weather data obtained from the example data source of Valencia

[29] https://github.com/smart-data-models/dataModel.Weather/tree/master/WeatherObserved



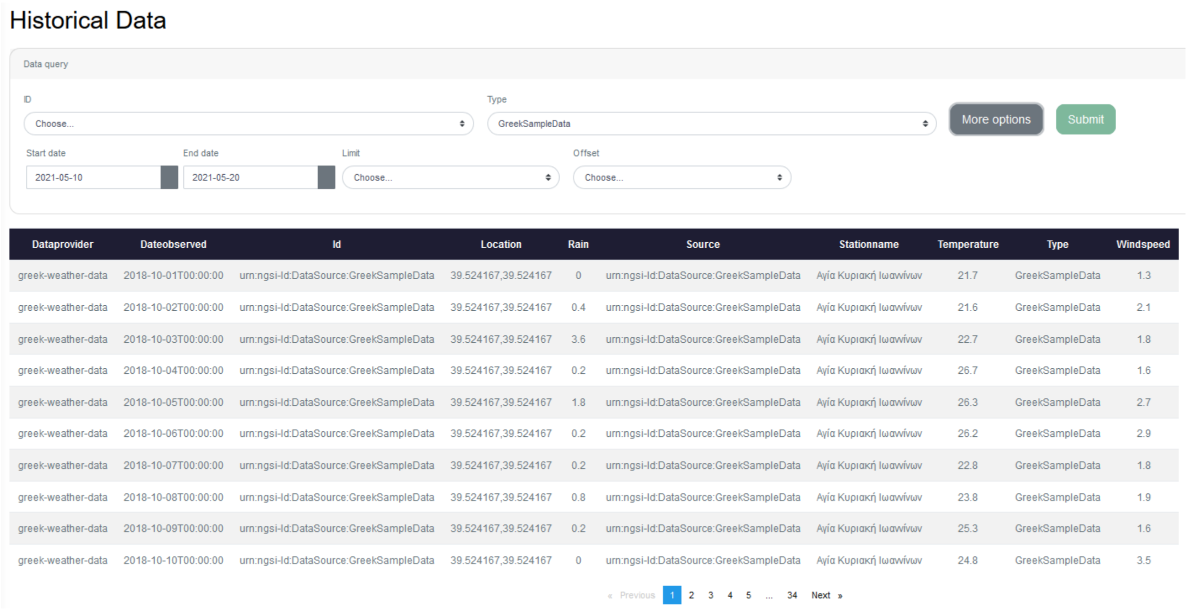
Historical Data

Data query

ID	Type	Start date	End date	Limit	Offset
Choose...	WeatherObserved	2021-05-10	2021-05-20	Choose...	Choose...

Dataprovider	Dateobserved	Humidity	Id	Location	Precipintensity	Precipprobability	Pressure	Source	Temperature	Type	Windspeed
https://api.darksky.net	2021-05-10T13:32:50Z	0.4	urn:DataSource:WeatherObserved	-0.377999,39.477167	0	0	1008.1	urn:DataSource:WeatherObserved	23.6	WeatherObserved	5.75
https://api.darksky.net	2021-05-10T13:32:50Z	0.4	urn:DataSource:WeatherObserved	-0.377999,39.477167	0	0	1008.1	urn:DataSource:WeatherObserved	23.6	WeatherObserved	5.75
https://api.darksky.net	2021-05-10T13:32:50Z	0.42	urn:DataSource:WeatherObserved	22.923216,40.63609	0	0	1020.2	urn:DataSource:WeatherObserved	22.93	WeatherObserved	4.43
https://api.darksky.net	2021-05-10T13:32:50Z	0.42	urn:DataSource:WeatherObserved	22.923216,40.63609	0	0	1020.2	urn:DataSource:WeatherObserved	22.93	WeatherObserved	4.43

Figure 32 – Combined historical weather data from two of the example data sources (Valencia and Thessaloniki)



Historical Data

Data query

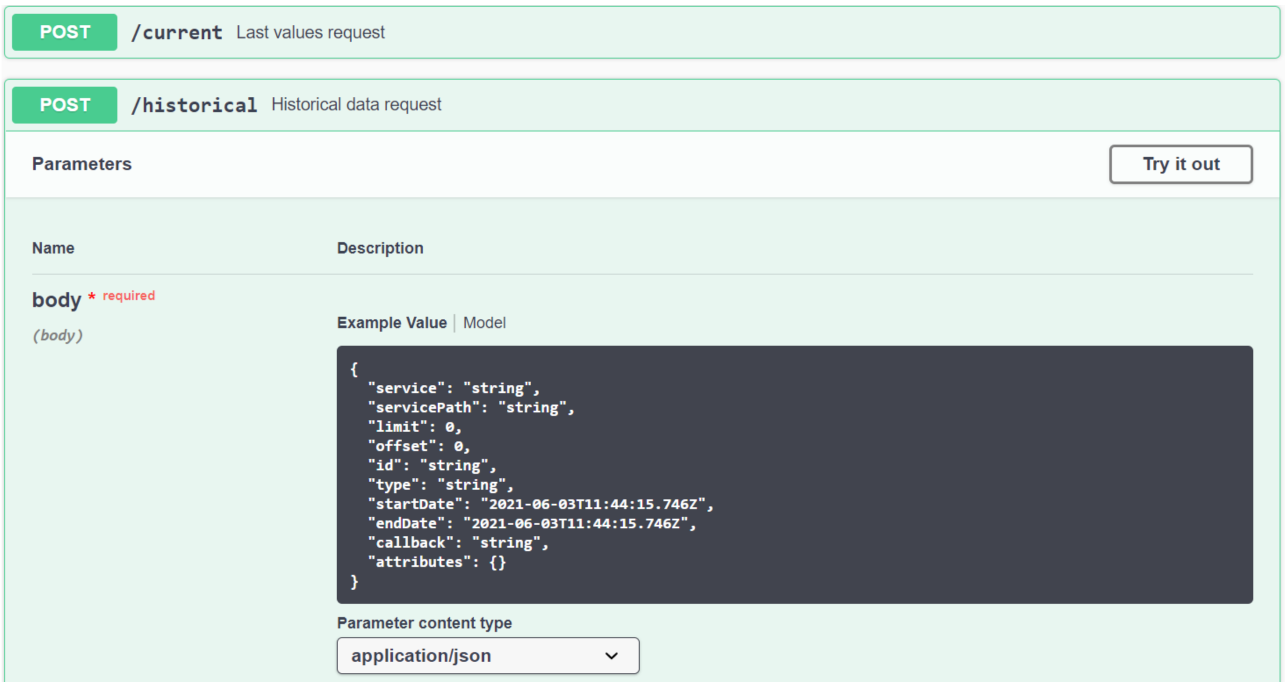
ID	Type	Start date	End date	Limit	Offset
Choose...	GreekSampleData	2021-05-10	2021-05-20	Choose...	Choose...

Dataprovider	Dateobserved	Id	Location	Rain	Source	Stationname	Temperature	Type	Windspeed
greek-weather-data	2018-10-01T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	21.7	GreekSampleData	1.3
greek-weather-data	2018-10-02T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0.4	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	21.6	GreekSampleData	2.1
greek-weather-data	2018-10-03T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	3.6	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	22.7	GreekSampleData	1.8
greek-weather-data	2018-10-04T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0.2	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	26.7	GreekSampleData	1.6
greek-weather-data	2018-10-05T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	1.8	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	26.3	GreekSampleData	2.7
greek-weather-data	2018-10-06T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0.2	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	26.2	GreekSampleData	2.9
greek-weather-data	2018-10-07T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0.2	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	22.8	GreekSampleData	1.8
greek-weather-data	2018-10-08T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0.8	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	23.8	GreekSampleData	1.9
greek-weather-data	2018-10-09T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0.2	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	25.3	GreekSampleData	1.6
greek-weather-data	2018-10-10T00:00:00	urn:ngsi-ld:DataSource:GreekSampleData	39.524167,39.524167	0	urn:ngsi-ld:DataSource:GreekSampleData	Αγία Κυριακή Ιωαννίνων	24.8	GreekSampleData	3.5

Figure 33 – Historical weather data obtained from an historical database registry of several locations in Greece

5.5.5 Scenario #4: Send information to the Data Abstraction and Virtualization component

The objective of this scenario is to illustrate how the DAV component can make use of the functionalities offered by the Semantic Interoperability component to retrieve data. Firstly, the list of available data sources and their corresponding agents can be obtained from the Metadata Registry, as has been shown in Section 5.2.6. Secondly, the DAV component can request the data to the On Demand component (Figure 34) through the API. The parameter "callback" indicates the URL where the data is going to be sent. Finally, the DAV component must be listening in the call-back port to receive the information. As an example, Figure 33 shows in a visual way the data which would be send to the DAV component. Then DAV performs its corresponding operations (Section 6) on the received data.


POST /current Last values request

POST /historical Historical data request

Parameters

Name | Description

body * required (body)

Example Value | Model

```
{
  "service": "string",
  "servicePath": "string",
  "limit": 0,
  "offset": 0,
  "id": "string",
  "type": "string",
  "startDate": "2021-06-03T11:44:15.746Z",
  "endDate": "2021-06-03T11:44:15.746Z",
  "callback": "string",
  "attributes": {}
}
```

Parameter content type: application/json

Figure 34 – On Demand component API

5.5.6 Scenario #5: Send information to the Analytics Services

The objective of this scenario is to illustrate how the Analytics Services can make use of the Semantic Interoperability Component to obtain their input data. To retrieve the new near real-time data via notifications, Analytics services subscribe to the data through the API. Firstly, the list of available data sources and their corresponding agents can be obtained from the Metadata Registry. Secondly, the Analytics service needs to define the parameters of the subscription and the endpoint to receive the notifications. Then, a subscription is available in the platform (it can be checked in the Admin UI) and waiting to send notifications. When an update in the Data Source is received, the Orion Context Broker sends a notification to the endpoint. This scenario is similar to Scenario #2 (Section 5.5.3), but in this case a Data Analytics service is who obtains the data from the Semantic Interoperability Component. It receives the notifications and processes them to perform its corresponding operations, described in deliverable D3.3 [15].

5.5.7 Scenario #6: Send Information to Dashboards, Services or Applications

In addition to the previous scenarios, the Semantic Interoperability Component API enables the creation of new applications that make use of the exposed functionality (Section 5.2.3). New applications could be developed to take advantage of the common access to the data and metadata and can be used in different premises. However, it is also possible to access the data making use of tools that involve zero programming effort, such as the mashup application tools, and to use their visualisation and processing widgets to provide added value services. For example, Fiware Wirecloud[30], which is an application mashup framework that supports NGSI data models, could be deployed on top of the Semantic Interoperability component. This would allow the creation of custom applications that use the data obtained from the connected data sources in simple and visual way. Figure 35 shows an instance of Wirecloud connected to the Semantic Interoperability API; the available widgets and operators are displayed, as well as an example of a widget obtaining data from a data source.

[30] https://wirecloud.readthedocs.io/en/stable/

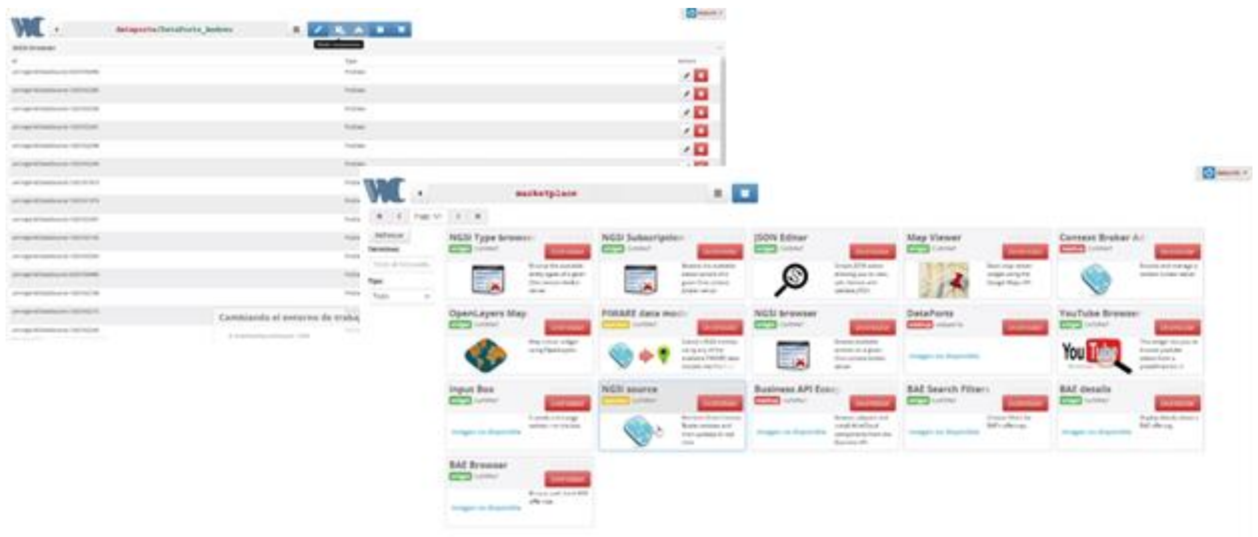


Figure 35 – Example of Wirecloud marketplace and a widget running using DataPorts data

Similarly, other existing tools that support NGSI, such as Node-RED[31], CKAN[32], Kibana[33], or Apache NiFi[34], could make use of the Semantic Interoperability API to retrieve data from the data sources, thus enabling a simple way to show the data and develop new services.

[31] https://nodered.org/

[32] https://fiware-ckan-extensions.readthedocs.io/en/latest/

[33] https://www.elastic.co/kibana

[34] https://nifi.apache.org/

6 DATA ABSTRACTION AND VIRTUALIZATION COMPONENT

This section covers the implementation status of the "Data Abstraction & Virtualization" component in M27 of the project. It provides an overview of the component as well as the involved technologies. The section also includes the description of the sub-components which compose the Data Abstraction & Virtualization (DAV) component, along with their technological aspects and their development status. Finally, Section 6.5 highlights the key functionalities of DAV via a running demonstration.

6.1 OVERVIEW

Data virtualization is a data integration technique that provides access to information via a virtualized service layer, regardless of the location of the data sources. It allows applications to access data, from a variety of heterogeneous sources, through a single endpoint, thus providing a unified, abstracted, and encapsulated view of information for query purposes, while being able to transform and process the data to prepare it for consumption. A significant challenge in data virtualization is to manage different types of storage systems (e.g., key-value, document, or relational databases) which all need to be integrated. In addition, data-intensive applications, that use a virtualized data source, still expect certain quality of service guarantees from the system, such as performance, availability, etc. The Data Abstraction and Virtualization (DAV) component attempts to deal with those challenges, also contributing to the data interoperability of the platform. Furthermore, it focuses on the fulfilment of the project's requirements that are related to quality of data. In a nutshell, DAV is responsible for correctly preparing data input from different sources inside the generic DataPorts architecture, maintaining metadata from all feeds, and finally making available the "cleaned" & processed datasets to any eventual client. Persistent data streams (that is, data that has already been collected and stored) are the primary source of load for the DAV.

6.2 TECHNOLOGICAL DESCRIPTION

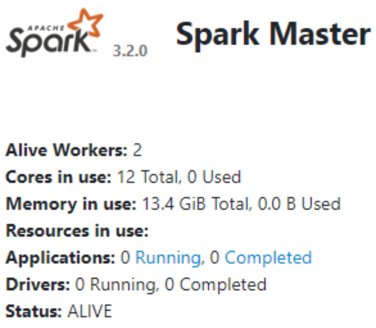
During the first reporting period of the project, emphasis was put on the design of the component, meaning to define its features and functionalities, according to the project's objectives and requirements. To this end, the internal architecture of DAV as well as its initial implementation were completed at that time and were documented in the deliverable D3.2 [16]. However, in the second phase of the project, the focus is mostly on selecting the proper technology to realize the component's functionalities and to adapt it to the other components' needs and characteristics, in order to enable the integration between the different layers of the DataPorts Platform. Towards that direction, a comparative analysis, concerning various Big Data processing frameworks was carried out, the conclusions of which are presented hereunder:

Given the project's requirements and constraints, the frameworks were analysed based on the following comparison metrics and parameters:

- computational performance in batch processing
- system distribution
- scalability
- compatibility with MongoDB
- compatibility with Kubernetes
- ability to combine information coming from many instances of DAV
- variety of exposed data formats

The frameworks taken under consideration were Apache Spark[35], Apache Flink[36], Apache Storm[37], and Apache Nifi[38]. These are all well-known tools, which are widely used in the Big Data industry.

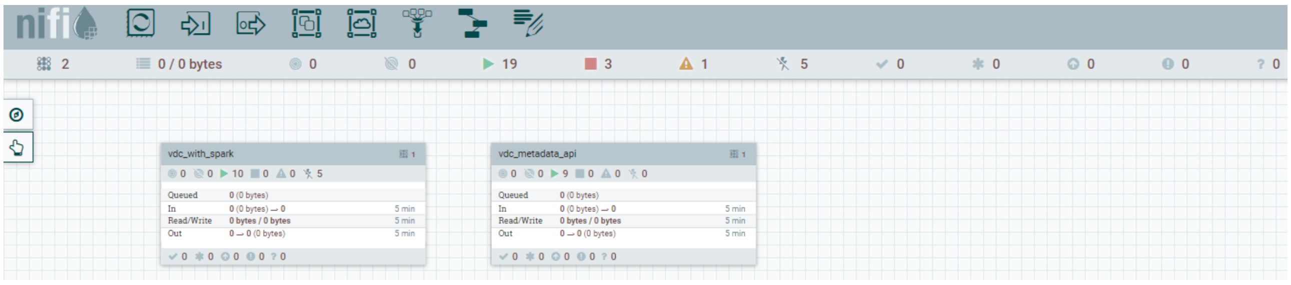
Concerning the data processing tasks of DAV, Apache Spark was chosen (Figure 36), since it meets all the above conditions. More specifically, it supports many programming languages (Java, Scala, Python, R), is well documented with a very wide community of users and can produce advanced analytical results. Compared to the other frameworks, it appears that each one has specific advantages and challenges. For example, Storm is mostly focused on processing real-time data [17], whilst DAV deals only with historical data. Regarding batch data processing, Spark and Flink both operate very well, each one outperforming the other depending on the benchmark (e.g., data type, size, job pattern etc.) [18][19]. However, Spark seems to be the framework with better scalability and overall faster runtimes [20]. For this reason, and for the richer analytical library and documentation, Spark was selected instead of Flink.



Alive Workers: 2
Cores in use: 12 Total, 0 Used
Memory in use: 13.4 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Figure 36 – Apache Spark cluster for the data processing tasks of DAV

As far as the exposure of the data is concerned, Apache Nifi was chosen (Figure 37). More specifically, this framework provides a virtual manipulation of data flows through a detailed and user-friendly UI, while its data flow programming philosophy has been an asset in both testing and production environments. It is worth noting that via Nifi, it is easy to create HTTP request handlers, which serve as REST API endpoints for triggering processing jobs. It is fully compatible with MongoDB, which makes the metadata extraction from DAV's repository very fast and efficient. It is also capable of transforming the data formats from JSON to Parquet or CSV etc. and combine other data sources to offer a variety of data to the consumers. Furthermore, Nifi is able to run in a cluster environment to avoid bottlenecks and reduce traffic, whereas it enables debugging at every step of the data flow. Moreover, it is very well documented offering many opportunities and tools. Finally, Apache Nifi is controlled via HTTPS protocols, thus making it safe and secure for production. Concerning its compatibility with Kubernetes, Apache Nifi is able to operate within a K8s cluster [21], so that can be integrated with the Virtual Data Repository, the DAV's data lake (Section 6.2.2).



vdc_with_spark 1

0 0 10 0 0 5

Queued	0 (0 bytes)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

0 0 0 0 ? 0

vdc_metadata_api 1

0 0 9 0 0 0

Queued	0 (0 bytes)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

0 0 0 0 ? 0

Figure 37 – Apache Nifi for DAV's interface

[35] https://spark.apache.org/

[36] https://flink.apache.org/

[37] https://storm.apache.org/

[38] https://nifi.apache.org/

The Data Abstraction & Virtualization component consists of three main sub-components, which communicate and interact, to fulfil DAV's objectives. The three sub-components, as shown in Figure 38, are:

- The (Pre)Processing and Filtering Software (PaFS)
- The Virtual Data Repository (VDR)
- The Virtual Data Container (VDC)

PaFS achieves the initial pre-processing (including cleaning and filtering) of the incoming datasets. VDR plays the role of the repository for all the existing datasets that have made their way in DAV, after being pre-processed by PaFS. It can be seen as a data lake, containing all the pre-processed datasets. VDC is the layer between DAV and any potential data recipient. It further processes the data, by applying any filtering rules that are defined by the data consumer per request. Furthermore, upon the dataset request by a recipient, it transforms the data into the desired format (e.g., Parquet, CSV, etc.). Moreover, it provides useful metadata for all the pre-processed datasets stored in VDR, such as size, number of rows and variables, timestamp of last update etc.

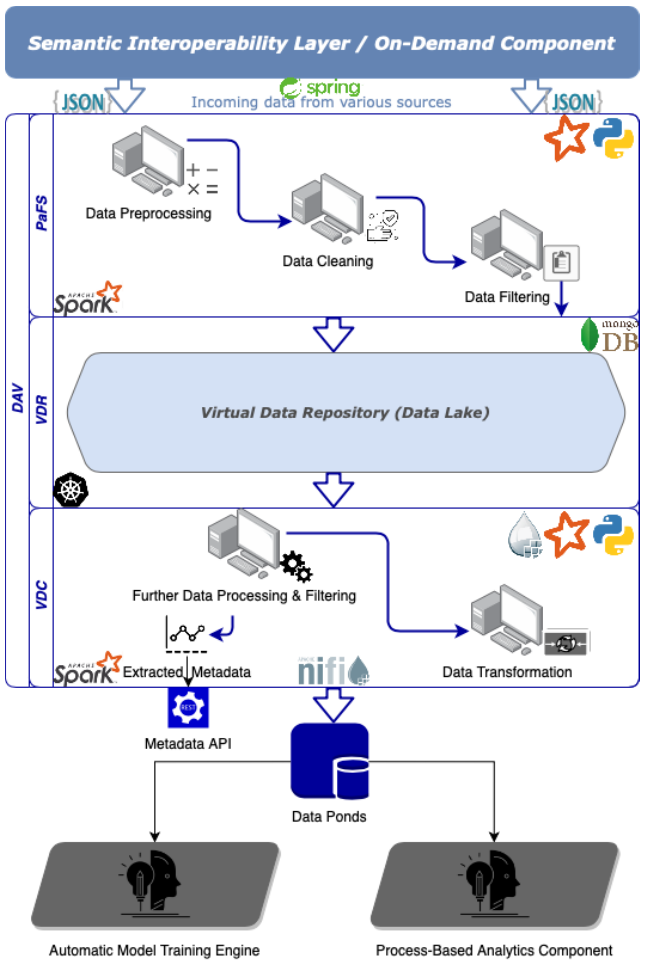

Semantic Interoperability Layer / On-Demand Component
{JSON}
spring
Incoming data from various sources
{JSON}
PaFS
Data Preprocessing
Data Cleaning
Data Filtering
Spark
mongo DB
DAV
VDR
Virtual Data Repository (Data Lake)
VDC
Further Data Processing & Filtering
Data Transformation
Spark
Extracted Metadata
nifi
Metadata API
Data Ponds
Automatic Model Training Engine
Process-Based Analytics Component

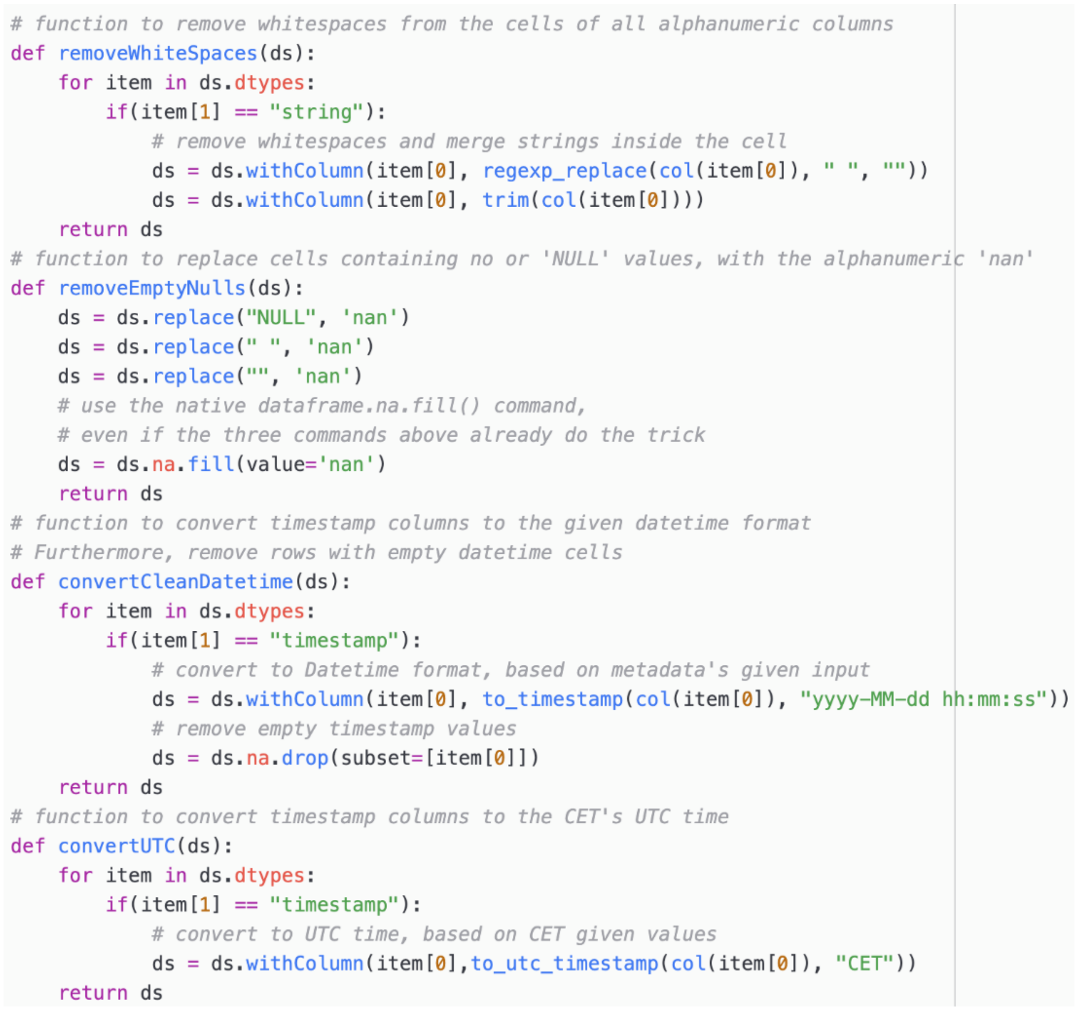
Figure 38 – Data Abstraction and Virtualization functionalities and technologies

6.2.1 (Pre)Processing & Filtering Software (PaFS)

Starting with PaFS, it is a sub-component that pre-processes the datasets coming from the on-demand component of the Semantic Interoperability layer upon request. All kinds of data can be accepted, since PaFS (and DAV in general) has a generic nature, regarding the data it receives and deals with. The incoming dataset is fully collected and, as soon as the collection ends (that is, as soon as the full dataset has been retrieved), PaFS creates a dataframe (a table), containing all the data. The proper column types for each dataset's column are examined, based on the dataset's metadata. If needed, PaFS acts by correcting some columns' data types. Such a step is highly important, since all later applications need proper column integrity within the dataset's frame. Then, the sub-component initiates the cleaning & filtering phase by proceeding to apply the following generic pre-processing techniques (to the dataset):

- Removal of whitespaces from all string-type cells
- Conversion of empty cells and 'NULL' string values to 'nan' values in all cells
- Removal of records / dataset rows without datetime values, or with wrong ones
- Conversion of datetime values to UTC format

These four steps were determined after thorough research and study upon the DataPorts Platform's needs for proper data cleaning & filtering. This way, future recipients of the cleaned datasets inside (and possibly outside) the platform can make the most out of their own frameworks, such as the Machine Learning tasks executed by the "Automatic Model Training Engine" [5] After that, PaFS locates all cells (belonging to any column) with no or 'NULL' value. These cells are filled with the alphanumeric 'nan', allowing future data recipients to easily remove these records (shall they choose to), with the use of simple existing - NaN removal - commands (available for many programming languages). PaFS's next move is to correct the datetime columns, meaning the ones of "timestamp" types. The sub-component locates these columns (if existent in the dataset) and proceeds to "conform" all data records to the predefined datetime format. This format is included in the dataset's metadata. This format is used by PaFS to correct all the other cells. For example, such a format would be "yyyy-MM-dd*HH:mm:ss:SSS". If the cells cannot be corrected based on the given format, or if they are empty, they (together with the dataset row to which they belong to) are removed. After the proper correction is complete, these timestamp columns are converted to UTC format, based on Valencia, Spain's (CET) time zone. To sum up, PaFS achieves the initial (but vital) pre-processing, cleaning and filtering of every incoming dataset. It pre-processes the dataset by the means of fully collecting it, transforming it into a Python code – friendly format and taking care of proper column – row structure. It cleans and filters the dataset by the means of applying specific general actions, generated based on DataPorts Platform's needs for proper function of the upper layer components (analytics). A portion of PaFS' code can be seen in Figure 39:



```
# function to remove whitespaces from the cells of all alphanumeric columns
def removeWhiteSpaces(ds):
    for item in ds.dtypes:
        if(item[1] == "string"):
            # remove whitespaces and merge strings inside the cell
            ds = ds.withColumn(item[0], regexp_replace(col(item[0]), " ", ""))
            ds = ds.withColumn(item[0], trim(col(item[0])))
    return ds
# function to replace cells containing no or 'NULL' values, with the alphanumeric 'nan'
def removeEmptyNulls(ds):
    ds = ds.replace("NULL", 'nan')
    ds = ds.replace(" ", 'nan')
    ds = ds.replace("", 'nan')
    # use the native dataframe.na.fill() command,
    # even if the three commands above already do the trick
    ds = ds.na.fill(value='nan')
    return ds
# function to convert timestamp columns to the given datetime format
# Furthermore, remove rows with empty datetime cells
def convertCleanDatetime(ds):
    for item in ds.dtypes:
        if(item[1] == "timestamp"):
            # convert to Datetime format, based on metadata's given input
            ds = ds.withColumn(item[0], to_timestamp(col(item[0]), "yyyy-MM-dd hh:mm:ss"))
            # remove empty timestamp values
            ds = ds.na.drop(subset=[item[0]])
    return ds
# function to convert timestamp columns to the CET's UTC time
def convertUTC(ds):
    for item in ds.dtypes:
        if(item[1] == "timestamp"):
            # convert to UTC time, based on CET given values
            ds = ds.withColumn(item[0],to_utc_timestamp(col(item[0]), "CET"))
    return ds
```

Figure 39 – PaFS code part of initial filtering & applying of the generic actions

Before finishing, PaFS proceeds to draw a correlation matrix between the columns of each dataset. This new piece of data is stored along with the existing (cleaned) dataset at the Virtual Data Repository (analysed later on). Furthermore, PaFS attempts to detect outliers in numerical columns, creating additional columns with cells that indicate which corresponding column cell is considered as an outlier. For every numerical column that is subject to outlier detection, one additional column is generated, with values of “yes” or “no”, indicating whether the corresponding cell is an outlier or not. The detection is implemented using the method of “three standard deviations from the mean”, as a cut-off radius / threshold. This method can be modified, based on the user’s & data recipient’s needs. PaFS main code is written in Python[39] programming language and, more specifically, Python Spark (PySpark) job. For the code to run as a Spark job in a Spark environment, some steps must be completed. Those steps are as follows:

- Make sure that an Apache Spark cluster has been set up on the physical machine(s) that PaFS will run upon
- Download all the necessary .jar files / libraries (available at DAV’s repository) and put them in the <user_path>/spark/jars folder
- Repeat the step above in all the nodes / workers of the Spark cluster. This is a useful step, since the user might need to change roles to their machines inside the cluster, using other masters and workers at times. Moving the PaFS PySpark job to the master every time, it shall work immediately, since the .jar files will be present

[39] https://www.python.org/

- Submit the PaFS's .py file as a Spark job to the master
- Check that PaFS is running, by examining its Spark job logs

To sum up, PaFS achieves the Pre-processing, Cleaning and Filtering of every incoming dataset. It pre-processes the dataset by the means of fully collecting it, transforming it into a Python code – friendly format and taking care of proper column – row structure. It cleans the dataset by the means of detecting "dirty" values (such as NaNs, empty fields, outliers, and wrong values). It filters the dataset by the means of eliminating all the dirty values found, either by replacing them, or by removing them (along with their rows in the dataset / dataframe).

6.2.2 Virtual Data Repository (VDR)

Moving on to the second main part of DAV, the Virtual Data Repository (VDR), this is the place where all the pre-processed, cleaned, and filtered datasets, coming from PaFS, are temporary saved. After a dataset has been passed through all its functions, it then gets stored in VDR (along with its columns' correlation matrix). VDR consists of a MongoDB[40], carrying modifications and custom parameters, to comply with DAV's efficiency standards. Apart from co-operating ideally with the Kubernetes framework, MongoDB is the most popular document store in 2020, according to DB-Engines[41], as already mentioned in the deliverable D2.1 [7]. In the case of DAV, MongoDB was also selected based on its auto-scaling / sharding capabilities, as well as its allowance of vital modifications / custom configurations by the creator. Since DAV's main functionality is data virtualization, MongoDB's co-existence with Kubernetes (as a container management tool) seems imperative. As depicted in Figure 38, VDR sub-component lives inside a Kubernetes cluster. Indeed, Kubernetes is the optimal solution and fits perfectly to the needs of DAV, since it achieves proper load balancing, whilst offering replication / scaling & scheduling techniques [22]. The version of Kubernetes upon which DAV is tested, is v1.20.5. The cluster consists of three physical machines (servers), running Ubuntu 20.04.2. There is one main node and two worker ones. For DAV to function, a Kubernetes Cluster must be set up & running in one (or more) physical machines. Then, it is preferable for DAV to be located inside a dedicated Kubernetes namespace.

An important note is that the following construction / implementation steps are for a local cluster system, with physical machines / servers. This means that these steps are not for an NFS server, or any implementation of this kind:

1. Make sure that Kubernetes points at DAV's namespace (as always)
2. Create a storageclass.yaml file, defining the information of the storage class that shall be used in the MongoDB – VDR
3. Define multiple persistent volumes, inside a persistentvolumes.yaml file. For that to work properly, we must have already created volumes at /mnt/disk/. Created volumes (such as vol1, vol2, vol3 etc.) inside the disk folder will be referenced through the .yaml file, as persistent volumes available for use from Mongo – VDR
   Note that there is no need to create persistent volume claims (PVCs), since these will be created automatically by the statefulSet.yaml file later on
4. Create a mongodb "headless" service .yaml file, for Mongo – VDR to function as a service inside the Kubernetes cluster

The last .yaml file needed is the statefulSet one. This is where the critical details of MongoDB's nature will be set, such as the number of replicas available after launch. A statefulSet.yaml example can be seen below, defining two Mongo replicas and an internal port to 27017:

[40] https://www.mongodb.com

[41] https://db-engines.com/en/ranking/document+store

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongo
spec:
  selector:
    matchLabels:
      app: mongo
  serviceName: "mongo"
  replicas: 2
  template:
    metadata:
      labels:
        app: mongo
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: mongo
        image: mongo
        command:
        - mongod
        - "--bind_ip_all"
        - "--replSet"
        - rs0
        ports:
        - containerPort: 27017
        volumeMounts:
        - name: volume-claim
          mountPath: /data/db
  volumeClaimTemplates:
  - metadata:
      name: volume-claim
    spec:
      storageClassName: managed-storage
      accessModes:
       - ReadWriteOnce
      resources:
        requests:
          storage: 3Gi
```

5. Apply all the .yaml files above, by the same order of creation, to the Kubernetes framework (kubectl create –f YAMLFILENAME)
6. Create a temporary mongo shell as a Kubernetes service, in order to enter the MongoDB created and configure it properly.

The result is the creation of a modified MongoDB, consisting of more than one replica, making it robust and immune to system fails, which of course depends on some vital factors, such as the number of replicas and their distribution inside the cluster. If a MongoDB replica goes down, the other replicas will ensure that VDR will still function as expected, containing all the data, and eliminating data loss or temporary unavailability. All Mongo Replicas are viewed as one database. Kubernetes performs load balancing and distributes data the way it thinks fits best, which means that the end user does not know where exactly (in which node or replica, for example) the queried response-data are stored.

6.2.3 Virtual Data Container (VDC)

Moving on to the third and final sub-component of DAV, the Virtual Data Container (VDC), this is the agent through which communication with data recipients is achieved, for data (stored in VDR) to be made available. It is a generic (sub) component, the role of which is to further process and filter the data, by applying specific filtering rules defined by the data consumers via HTTP POST requests. The scope of those rules is twofold: i) on one hand to filter the datasets and thus to serve only the specific data pond that a specific user is interested in and ii) on the other hand to detect and remove wrong inputs (from areas / columns where those inputs are not acceptable e.g., outdoor temperatures at minus 100 degrees Celsius), which are most probably caused by sensors malfunction. Through that POST request, the data consumer also defines the format in which he/she wants to receive the data (data transformation). Furthermore, VDC is responsible for exposing useful metadata (size, number of rows and variables, timestamp of last update etc.) for each one dataset that is stored in VDR. Those metadata are available via a RESTful API.

<u>Interface</u>

As already mentioned in Section 6.2, Apache Nifi is used to create the data flows, for DAV to be able to expose the cleaned and processed data to the data consumers. To this end, Figure 40 presents the flow developed to implement the VDC Rules System (data processing and filtering based on user-defined rules), which is analysed in the following subsection. As can be seen in that figure, the actual processing, filtering, and transformation of the data is performed by Spark. The following bullets briefly describe the functionality that is executed by each one part of the depicted flow:

- VDC API -> The HTTP POST endpoint for the VDC API
- PARSE JSON REQUEST DATA AND GENERATE SPARK JOB SPECIFICATIONS -> Parse the input data (rules file, desired output data format, etc.) and create the corresponding Spark Job request
- INVOKE SPARK JOB AND SEND ACK -> Send the request to the Spark Listener and also send an acknowledgement message to the user who made the request
- HANDLING ERRORS -> Generate an HTTP response in case something goes wrong during the processing of the input data (e.g., errors in the structure of the rules file)

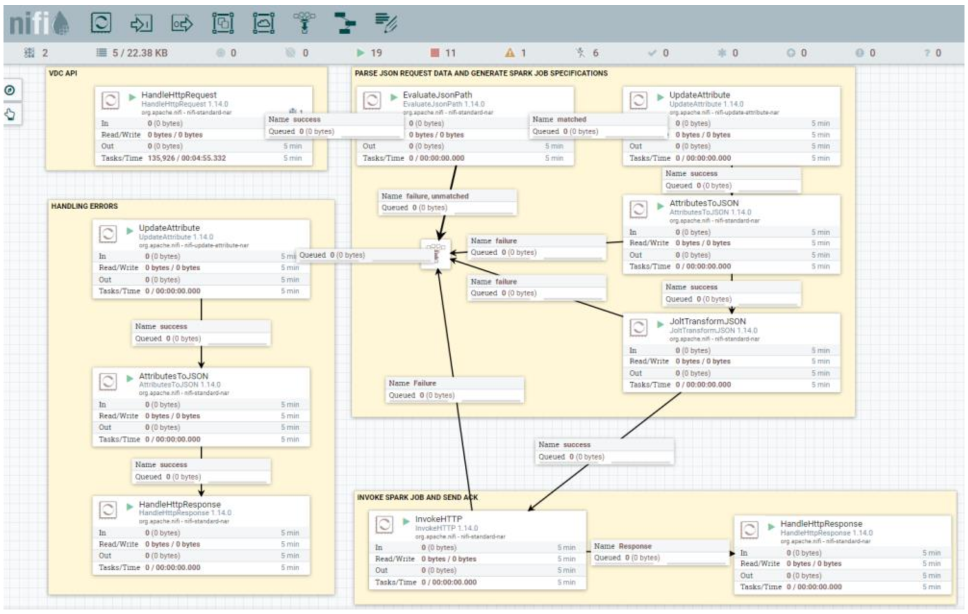


Figure 40 – VDC flow to trigger data processing, filtering and transformation jobs upon request

Correspondingly, Figure 41 shows the metadata extraction flow. The different parts of the flow implement the following functionalities:

- METADATA API LISTENER -> The HTTP GET endpoint for the datasets' metadata API
- COMMUNICATING WITH MongoDB -> Run the appropriate MongoDB queries for collecting the datasets' statistics
- PARSING DATASETS METADATA AND SENDING AS JSON RESPONSE -> Parse the collected metadata into the format of the API specification and send an HTTP response
- HANDLING ERRORS -> Generate an HTTP response in case something goes wrong during the execution (runtime errors such as collection or DB not found)

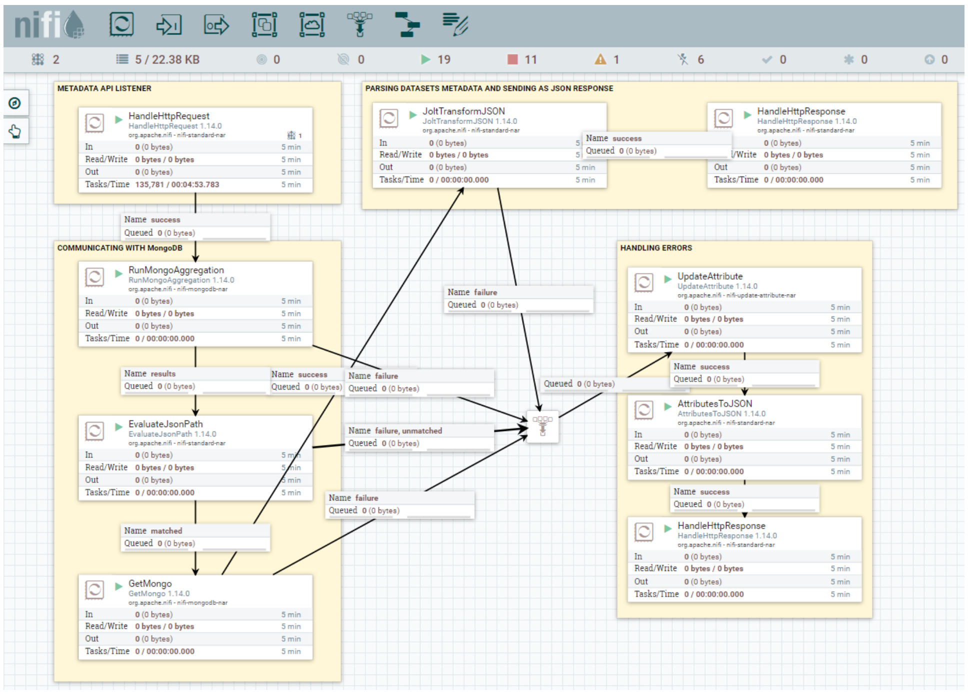


Figure 41 – VDC flow for metadata extraction upon request

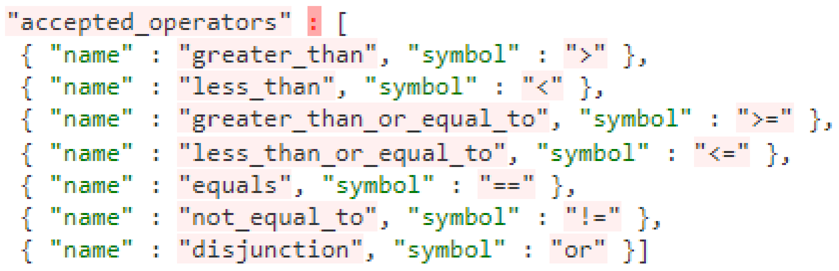
VDC Rules System

This subsection provides the structure of the aforementioned filtering rules, written in JSON format, as well as some indicative examples, related to the Valencia PCS traffic dataset.

The rule structure is very simple. The three core elements of each rule are a "subject column", an "operator" and the "object". The expected rules list format is a JSON Array, which shall include rules (JSON Objects), containing those string values. VDC parses this list and applies the rules to the requested dataset.

This is the architecture of the incoming rules JSON file:

- A JSON Object, containing
  - A string field with the dataset's name, and another one with the dataset's id
  - A JSON array containing the rules as JSON Objects
    - Each JSON Object (rule) in the array shall contain a string field with its name, and another JSON Object with the rule itself
      - Each rule Object shall include the “subject_column”, “operator” and “object” fields
      - In case the “operator” is a disjunction, meaning the “or” expression, then the “object” field shall be a JSON Array, containing two (or more) objects with single string “operator” and “object” fields



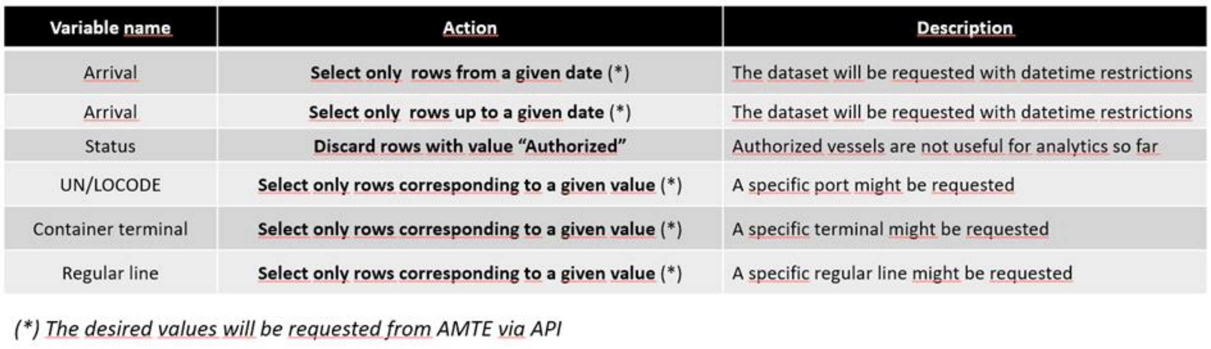
```
"accepted_operators" : [
 { "name" : "greater_than", "symbol" : ">" },
 { "name" : "less_than", "symbol" : "<" },
 { "name" : "greater_than_or_equal_to", "symbol" : ">=" },
 { "name" : "less_than_or_equal_to", "symbol" : "<=" },
 { "name" : "equals", "symbol" : "==" },
 { "name" : "not_equal_to", "symbol" : "!=" },
 { "name" : "disjunction", "symbol" : "or" }]
```

Figure 42 – VDC rules system accepted operators

Figure 42 indicates all the accepted operators that can be used by the rules' author (data scientist, application developer, end-user etc.). Any other operator is not recognized by the VDC. Therefore, any rule object containing unknown operators is discarded. In addition, there are two main principles, on which the rules system is based upon. These principles enable a user to better understand the rules' nature:

- A rule's main goal is to apply filters to one subject (column) at a time, not combining subjects (columns). If more than one column is concerned as subjects in one rule, then this step could be regarded more as "pre-processing" and less as "filtering". Moreover, changing the content of specific rows / values, or removing rows with specific value type, is also a step at lower level than what the rules' system suggest
- DAV is, from its nature, a generic framework, meaning that it can be used for all kinds of datasets. Making a rule more complex than that of the standard "subject - operator - object" architecture, simply violates DAV's generic nature. It is relatively easy and simple to implement basic pre-processing steps for selected datasets; it usually does not take more than - a couple of - lines of code. However, these kinds of pre-processing steps would not be applied to other datasets, assuming that any kind of dataset can enter DAV. Therefore, the data scientists would have to go to a conditional solution, such as "if the incoming dataset is X, then apply these selected lines of code". This is a very easy solution, but ruins DAV's fundamental generic nature

An indicative example of the rules' system usability is derived from the specific filtering actions that were requested from the Automatic Model Training Engine to be applied to the Valencia PCS traffic dataset. Those actions are presented in Figure 43:

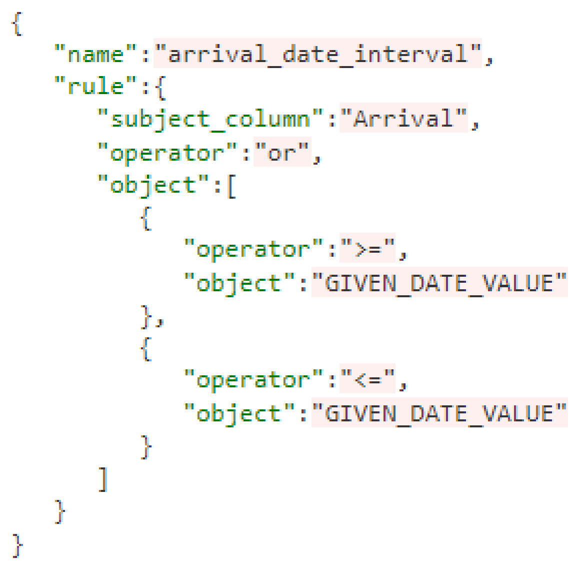


Variable name	Action	Description
Arrival	**Select only rows from a given date** (*)	The dataset will be requested with datetime restrictions
Arrival	**Select only rows up to a given date** (*)	The dataset will be requested with datetime restrictions
Status	**Discard rows with value "Authorized"**	Authorized vessels are not useful for analytics so far
UN/LOCODE	**Select only rows corresponding to a given value** (*)	A specific port might be requested
Container terminal	**Select only rows corresponding to a given value** (*)	A specific terminal might be requested
Regular line	**Select only rows corresponding to a given value** (*)	A specific regular line might be requested

(*) The desired values will be requested from AMTE via API

Figure 43 – Filtering actions requested by AMTE for the Valencia PCS traffic dataset

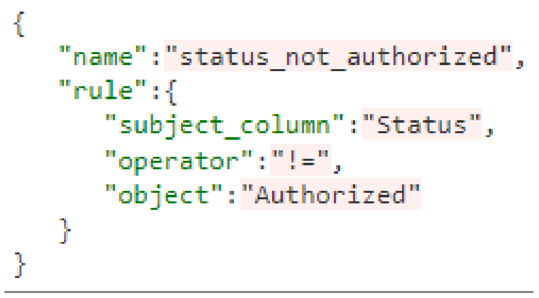
Regarding the first two actions for the "Arrival" variable (column), these should be implemented as a rule with disjunction, meaning "or". The JSON object of this rule (containing the two "Arrival" column actions) should be written according to the following example:



```
{
   "name":"arrival_date_interval",
   "rule":{
      "subject_column":"Arrival",
      "operator":"or",
      "object":[
         {
            "operator":">=",
            "object":"GIVEN_DATE_VALUE"
         },
         {
            "operator":"<=",
            "object":"GIVEN_DATE_VALUE"
         }
      ]
   }
}
```

Figure 44 – Filtering rules JSON file example #1

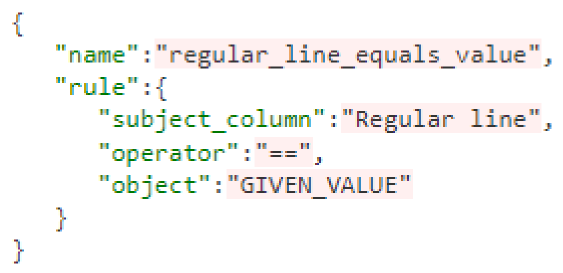
Regarding the third action, related to the “Status” variable (column), the rule’s author should use the “not equal to” operator, meaning “!=“. The JSON object of this rule should be written according to the following example:



```
{
   "name":"status_not_authorized",
   "rule":{
      "subject_column":"Status",
      "operator":"!=",
      "object":"Authorized"
   }
}
```

Figure 45 – Filtering rules JSON file example #2

All three remaining actions are related to the “equal to” operator, meaning “==“. Therefore, the proper way to write down the corresponding rules JSON file should be (taking the “Regular line” column as an example):

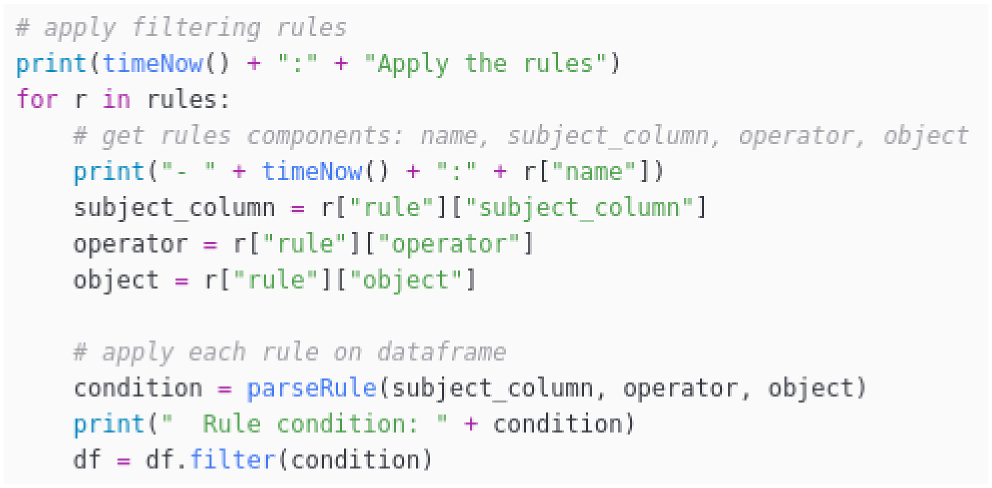


```
{
   "name":"regular_line_equals_value",
   "rule":{
      "subject_column":"Regular line",
      "operator":"==",
      "object":"GIVEN_VALUE"
   }
}
```

Figure 46 – Filtering rules JSON file example #3

In conclusion, the goal is to assist any potential DataPorts user to define a JSON Array of a dataset’s actions / rules with ease. For that reason, its structure is kept simple, and the principles behind the rules’ architecture are sane. Consequently, those filtering rules can be specified not only by data scientists, but also by users with limited or no technical background within a ports ecosystem, who are interested in exploiting DataPorts cognitive services. The rules system can be applied to all kinds of datasets, based on the needs for cleaning & filtering.

The VDC Rules System is implemented in Python programming language with the usage of Spark framework. Part of the corresponding code is depicted in Figure 47:



```
# apply filtering rules
print(timeNow() + ":" + "Apply the rules")
for r in rules:
    # get rules components: name, subject_column, operator, object
    print("- " + timeNow() + ":" + r["name"])
    subject_column = r["rule"]["subject_column"]
    operator = r["rule"]["operator"]
    object = r["rule"]["object"]

    # apply each rule on dataframe
    condition = parseRule(subject_column, operator, object)
    print("  Rule condition: " + condition)
    df = df.filter(condition)
```

Figure 47 – VDC code part of applying the filtering rules

6.3 DEVELOPMENT STATUS (M27)

This section provides the features (sorted by subcomponent) of DAV that are currently implemented (please refer to Figure 38).

- Pre-processing & Filtering Software (PaFS)
    - Main functionalities completed
        - Pre-processing
        - Cleaning
        - Filtering
        - Correlation
        - Outliers detection
    - Requirements that are currently addressed (partially or totally) by PaFS
        - R3.19: The Data Abstraction and Virtualization component must be data-source independent
        - R3.41: The DataPorts Platform should have (efficient) provisions for checking data quality (e.g., to detect concept drift, missing data, inconsistent data etc.)
        - R3.42: The DataPorts Platform should deliver cleansed, integrated etc. data to the analytics services
        - R3.44: Data correlation. The data from different sources should be correlated (virtual object) or in the same process
        - R3.46: Data quality. Before correlate data from different sources, quality should be checked to guarantee the correct interpretation
- Virtual Data Repository (VDR)
    - Infrastructure implemented (Data Lake)
        - MongoDB Replicas up & running in a Kubernetes cluster
    - Requirements and functionalities that are currently fulfilled (partially or totally) by VDR
        - R3.19: The Data Abstraction and Virtualization component must be data-source independent
        - R3.23: The components of the DataPorts Platform could be virtualized, in order to ease its deployment and portability
        - F2.2: The DataPorts Platform sets a data driven ecosystem ready for a comprehensive exploitation of data, and virtual data repositories
        - F3.10: The DataPorts Platform provides declarative, distributed data aggregation

- Virtual Data Container (VDC)
    - Main functionalities completed
        - An abstraction layer between the data consumers (e.g., data-intensive applications) and the VDR
        - Further processing and filtering based on rules upon request
        - Returns a portion of the data (data pond) upon request
        - Able to transform the data to serve it in the right format (e.g., from JSON to Parquet, CSV, etc.)
        - Provide metadata for all the pre-processed datasets stored in VDR
    - Requirements and functionalities that are currently fulfilled (partially or totally) by VDC
        - R3.18: The Data Abstraction and Virtualization component must have an open API for big vendors as well as new providers to be able to publish their services and components
        - F3.15: The DataPorts Platform provides smart API for cognitive services

6.4 FUTURE WORK: ROADMAP

Towards the finalization of the project, emphasis will be put on making all the necessary adjustments related to the integration of Data Abstraction & Virtualization component into the DataPorts Platform, from the use cases perspective. For example, additional pre-processing actions or new filtering rules files might need to be created, depending on the available datasets as well as the end-users' requirements. Furthermore, effort will be spent on how to package the component, so that can be easily deployed in a common infrastructure inside the ports premises if needed. A lightweight version of DAV might be developed for that purpose. Regarding the component itself, the goal is to further improve VDR, to become a fully dynamic system for storing and managing Big Data, which could automatically scale its resources according to the network load, aiming at optimizing resource utilization and decreasing response times for applications in need of high availability. DAV component will be released as open-source software under GNU General Public License version 3.

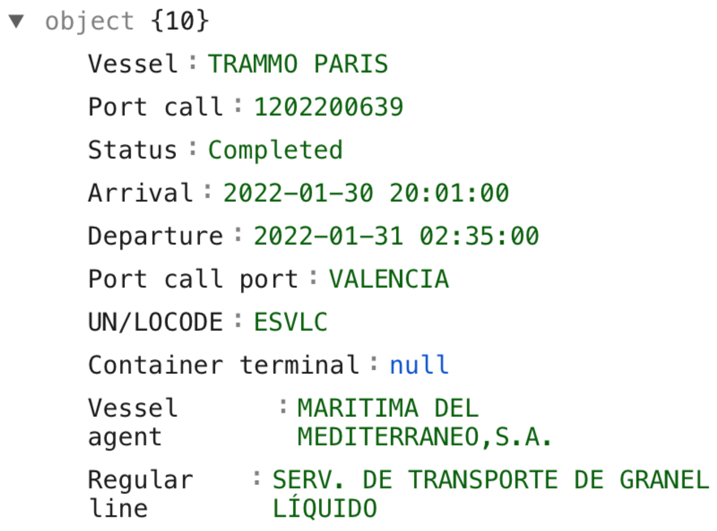
6.5 EXAMPLE OF USE: DEMONSTRATION

6.5.1 Introduction

Data Abstraction & Virtualization component is considered as an internal framework of the DataPorts Platform, thus users interact with VDC (DAV's third sub-component) mainly via exposed RESTful APIs. The following running demonstrations highlight the DAV's main functionalities, already described in the previous subsections. **The PCS traffic dataset from Valencia port** is used for demo purposes. This dataset is free to use and can be accessed here:
https://www.valenciaportpcs.net/portcalls/search/historic

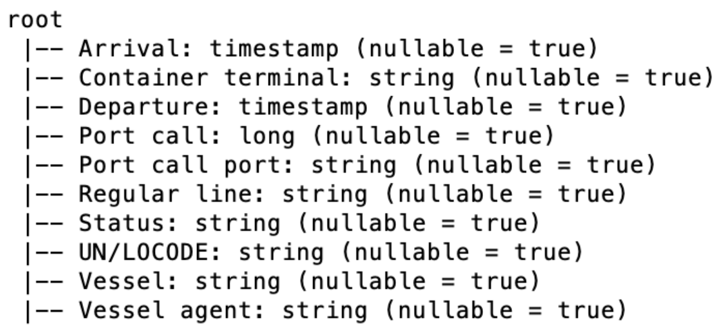
6.5.2 Pre-processing, Cleaning and Filtering

Figure 48 depicts an example of those data coming from the Semantic Interoperability layer in JSON format, while Figure 49 lists the datatype for each one attribute. These metadata are necessary for PaFS (DAV's first sub-component) to work properly, as explained in Section 6.2.1. PaFS is then responsible for performing all the generic pre-processing techniques that are described in Section 6.2.1, resulting in the pre-processed and cleaned data, presented in Figure 50:



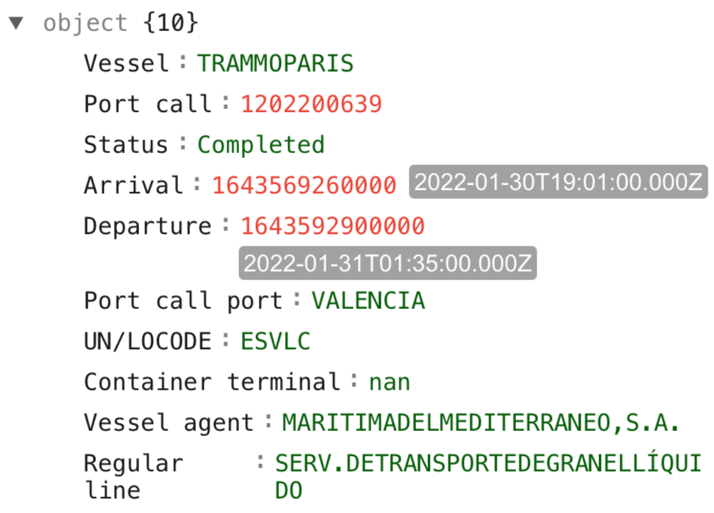
```
▼ object {10}
    Vessel : TRAMMO PARIS
    Port call : 1202200639
    Status : Completed
    Arrival : 2022-01-30 20:01:00
    Departure : 2022-01-31 02:35:00
    Port call port : VALENCIA
    UN/LOCODE : ESVLC
    Container terminal : null
    Vessel      : MARITIMA DEL
    agent         MEDITERRANEO,S.A.
    Regular    : SERV. DE TRANSPORTE DE GRANEL
    line         LÍQUIDO
```

Figure 48 – Valencia PCS traffic raw data example



```
root
 |-- Arrival: timestamp (nullable = true)
 |-- Container terminal: string (nullable = true)
 |-- Departure: timestamp (nullable = true)
 |-- Port call: long (nullable = true)
 |-- Port call port: string (nullable = true)
 |-- Regular line: string (nullable = true)
 |-- Status: string (nullable = true)
 |-- UN/LOCODE: string (nullable = true)
 |-- Vessel: string (nullable = true)
 |-- Vessel agent: string (nullable = true)
```

Figure 49 – Valencia PCS traffic datatypes (Spark schema)



```
▼ object {10}
    Vessel : TRAMMOPARIS
    Port call : 1202200639
    Status : Completed
    Arrival : 1643569260000 2022-01-30T19:01:00.000Z
    Departure : 1643592900000
                2022-01-31T01:35:00.000Z
    Port call port : VALENCIA
    UN/LOCODE : ESVLC
    Container terminal : nan
    Vessel agent : MARITIMADELMEDITERRANEO,S.A.
    Regular    : SERV.DETRANSPORTEDEGRANELLÍQUI
    line         DO
```

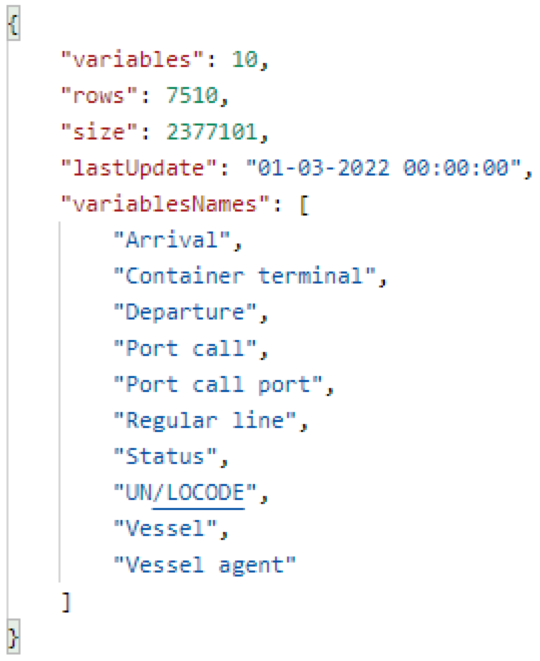
Figure 50 – Valencia PCS traffic pre-processed and cleaned data example

As shown in Figure 50:

- whitespaces from all string-type cells have been removed
- ‘NULL’ string values have been converted to ‘nan’
- CET timestamps have been converted to UTC

6.5.3 Request VDC to extract metadata of a dataset stored in VDR

Given that the above pre-processed and cleaned dataset has been stored properly in VDR, a potential data consumer is then able to extract its metadata through an HTTP GET request, as shown in Figure 51. The id of the dataset is passed to the VDC as a query parameter. Postman [42]tool is used to send that request:



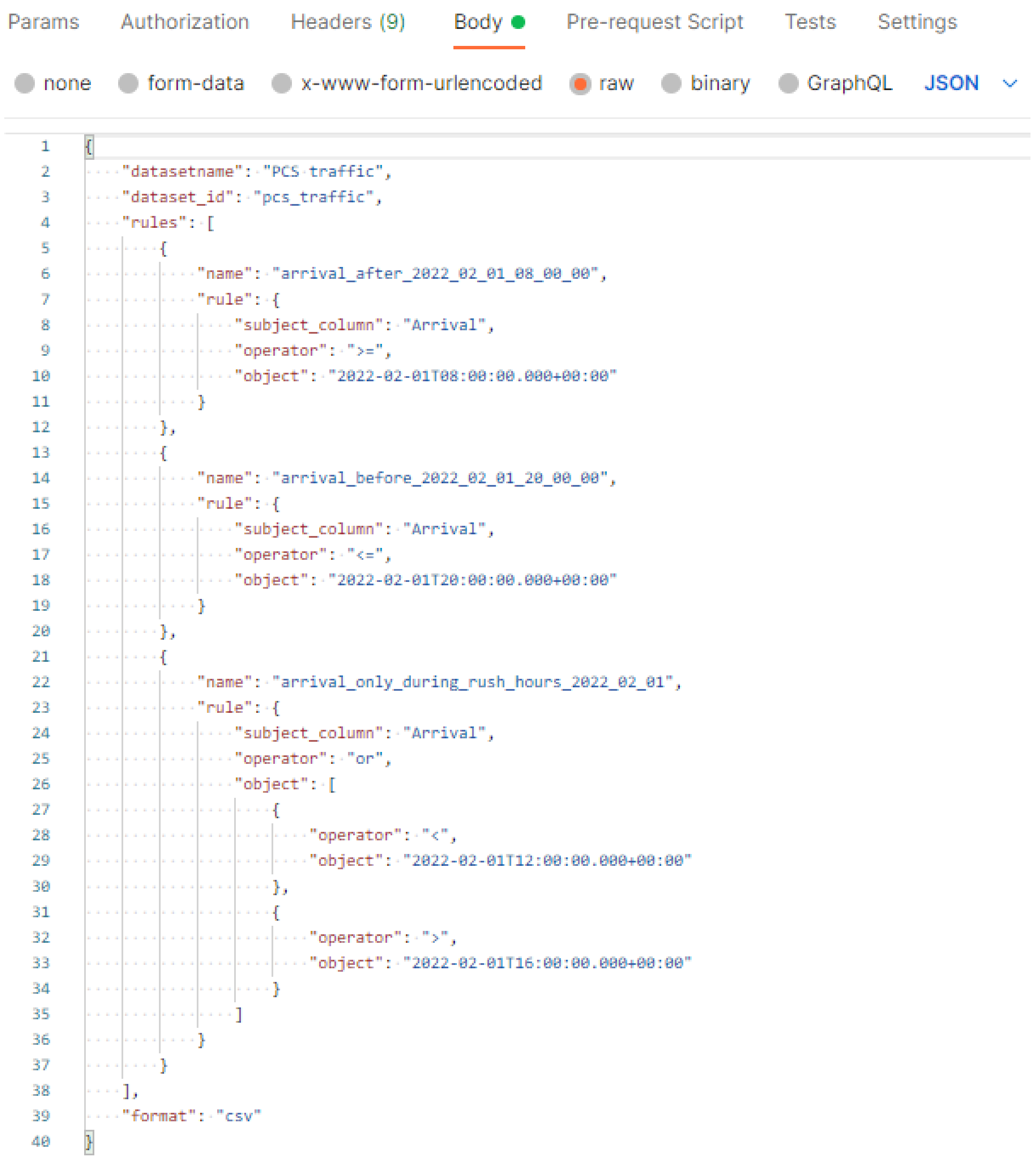
```
{
    "variables": 10,
    "rows": 7510,
    "size": 2377101,
    "lastUpdate": "01-03-2022 00:00:00",
    "variablesNames": [
        "Arrival",
        "Container terminal",
        "Departure",
        "Port call",
        "Port call port",
        "Regular line",
        "Status",
        "UN/LOCODE",
        "Vessel",
        "Vessel agent"
    ]
}
```

Figure 51 – VDC metadata extraction example

6.5.4 Request VDC to retrieve a data pond (via filtering rules) in a specific format

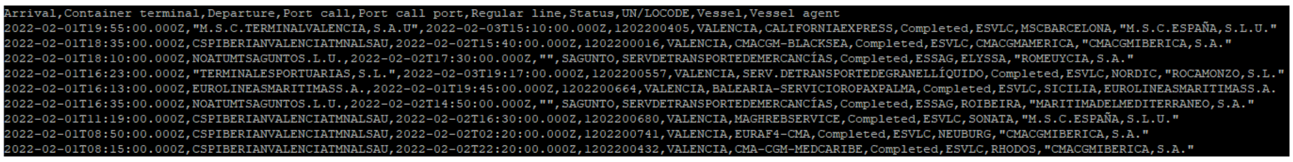
Given that the same pre-processed and cleaned dataset has been stored properly in VDR, a potential data consumer is then able to retrieve data ponds of that dataset through an HTTP POST request. For example, a user might want to obtain only the PCS traffic data that refer to specific time intervals (e.g., during rush hours of a specific day), and also that data pond to be structured in CSV format. Figure 52 depicts the corresponding body of the POST request, while Figure 53 presents the output filtered data. Again, Postman is used to send the request.

[42] https://www.postman.com/



```json
{
    "datasetname": "PCS traffic",
    "dataset_id": "pcs_traffic",
    "rules": [
        {
            "name": "arrival_after_2022_02_01_08_00_00",
            "rule": {
                "subject_column": "Arrival",
                "operator": ">=",
                "object": "2022-02-01T08:00:00.000+00:00"
            }
        },
        {
            "name": "arrival_before_2022_02_01_20_00_00",
            "rule": {
                "subject_column": "Arrival",
                "operator": "<=",
                "object": "2022-02-01T20:00:00.000+00:00"
            }
        },
        {
            "name": "arrival_only_during_rush_hours_2022_02_01",
            "rule": {
                "subject_column": "Arrival",
                "operator": "or",
                "object": [
                    {
                        "operator": "<",
                        "object": "2022-02-01T12:00:00.000+00:00"
                    },
                    {
                        "operator": ">",
                        "object": "2022-02-01T16:00:00.000+00:00"
                    }
                ]
            }
        }
    ],
    "format": "csv"
}
```

Figure 52 – VDC request body example



```
Arrival,Container terminal,Departure,Port call,Port call port,Regular line,Status,UN/LOCODE,Vessel,Vessel agent
2022-02-01T19:55:00.000Z,"M.S.C.TERMINALVALENCIA,S.A.U",2022-02-03T15:10:00.000Z,1202200405,VALENCIA,CALIFORNIAEXPRESS,Completed,ESVLC,MSCBARCELONA,"M.S.C.ESPAÑA,S.L.U."
2022-02-01T18:35:00.000Z,CSPIBERIANVALENCIATMNALSAU,2022-02-02T15:40:00.000Z,1202200016,VALENCIA,CMACGM-BLACKSEA,Completed,ESVLC,CMACGMAMERICA,"CMACGMIBERICA,S.A."
2022-02-01T18:10:00.000Z,NOATUMTSAGUNTOS.L.U.,2022-02-02T17:30:00.000Z,"",SAGUNTO,SERVDETRANSPORTEDEMERCANCÍAS,Completed,ESSAG,ELYSSA,"ROMEUYCIA,S.A."
2022-02-01T16:23:00.000Z,"TERMINALESPORTUARIAS,S.L.",2022-02-03T19:17:00.000Z,1202200557,VALENCIA,SERV.DETRANSPORTEDEGRANELLÍQUIDO,Completed,ESVLC,NORDIC,"ROCAMONZO,S.L."
2022-02-01T16:13:00.000Z,EUROLINEASMARITIMASS.A.,2022-02-01T19:45:00.000Z,1202200664,VALENCIA,BALEARIA-SERVICIOROPAXPALMA,Completed,ESVLC,SICILIA,EUROLINEASMARITIMASS.A.
2022-02-01T16:35:00.000Z,NOATUMTSAGUNTOS.L.U.,2022-02-02T14:50:00.000Z,"",SAGUNTO,SERVDETRANSPORTEDEMERCANCÍAS,Completed,ESSAG,ROIBEIRA,"MARITIMADELMEDITERRANEO,S.A."
2022-02-01T11:19:00.000Z,CSPIBERIANVALENCIATMNALSAU,2022-02-02T16:30:00.000Z,1202200680,VALENCIA,MAGHREBSERVICE,Completed,ESVLC,SONATA,"M.S.C.ESPAÑA,S.L.U."
2022-02-01T08:50:00.000Z,CSPIBERIANVALENCIATMNALSAU,2022-02-02T02:20:00.000Z,1202200741,VALENCIA,EURAF4-CMA,Completed,ESVLC,NEUBURG,"CMACGMIBERICA,S.A."
2022-02-01T08:15:00.000Z,CSPIBERIANVALENCIATMNALSAU,2022-02-02T22:20:00.000Z,1202200432,VALENCIA,CMA-CGM-MEDCARIBE,Completed,ESVLC,RHODOS,"CMACGMIBERICA,S.A."
```

Figure 53 – VDC output data pond example

7 DATA PROCESSING SERVICES COMMON EXAMPLE OF USE

The Data Processing Services are the components that take the role of the functional connection between the heterogeneous data sources available with the data analytics services and cognitive applications of the platform. On the one hand, the project is accessing a considerable number of heterogeneous data sources via the Data Access Component (explained in D3.1 [2]). On the other hand, DataPorts is producing several Data Analytics Services and Cognitive Applications (explained in D3.6 [5]). The whole DataPorts Platforms is going to be validated with the scenarios and data of pilot use cases (covered by WP5). However, to plan an organized and effective integration process some scenarios are offered in this section. They are defined taking as starting point the access to several data sources selected with the aim to describe and verify the feasibility and operation of these DataPorts Platform components working together.

In addition, each component of the Data Processing Services has been presented independently in Sections 5 and 6. In addition, the improvement of the Data Access component since M18 of the project has been described in Section 4. For each of those components, its functionalities, technical decisions taken for its implementation and a description of its operation have been explained. However, as Section 3 has stated, the integration will be one of the main challenges of the release of the DataPorts platform. The integration has taken place following the general platform integration plan, which ends in the M30. In this section, the status of the platform (M27) is presented through a prototype that has been developed to review and analyse the integration of the data processing services and relevant analytics services.

Finally, it is crucial to highlight that during M30 the final status of the DataPorts Platform Implementation and Integration will be provided in the platform portal and Git repositories (listed in Section 8). This final phase includes all the activities aimed at the implementation of the data platform, considering all the technical aspects including those related to data governance. This phase is carried out through the ending of WP2, WP3 and WP4 and the results of the D3.7 -Permissioned Blockchain network [4], D4.4 –Blockchain-based data governance rules and D4.6 -Secure environment for data sharing and trading.

7.1 DEMONSTRATION OVERVIEW

Currently, the data platform implementation and services are providing the blocks included in the architecture of the data platform, covering all the aspects of the data value chain, with a special focus on data integration and aggregation. For that reason, this demo is devoted to explaining the interactions of the core functional components of the DataPorts platform. Mainly, it is focused on explaining the technical tools for the acquisition, interoperation, processing, and analysis of the data coming from heterogeneous data sources available in the seaports' ecosystem. As Figure 54 shows, the common demonstration is structured in three main blocks, firstly the Data Access Component (explained in deliverable D3.1 [2] and updated in this deliverable), secondly, the Data Processing Services (explained in this deliverable) and, thirdly, the Data Analytics Services and Cognitive Applications (explained in deliverable D3.6 [5]).

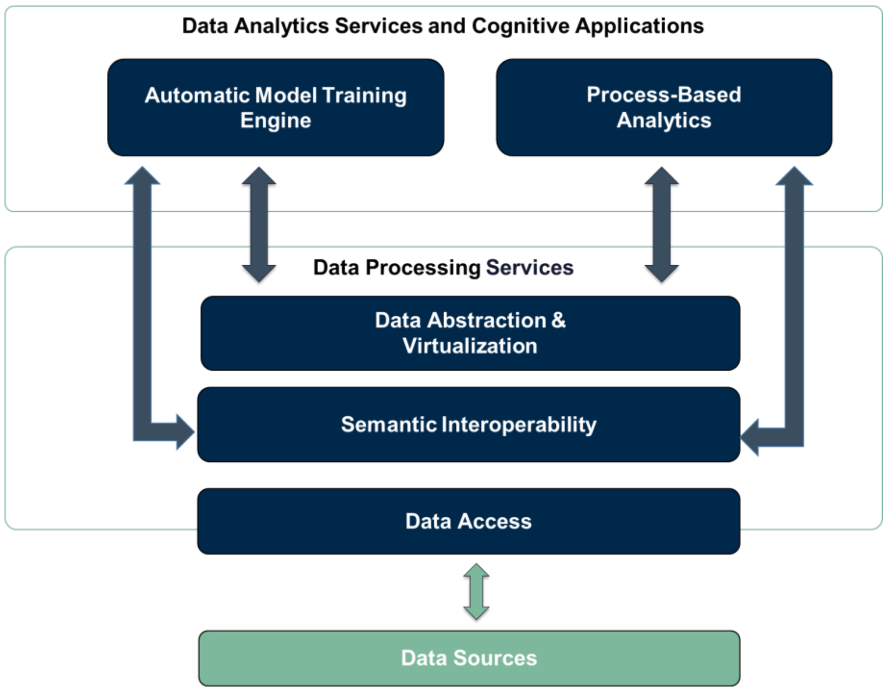

Data Analytics Services and Cognitive Applications
Automatic Model Training Engine
Process-Based Analytics
Data Processing Services
Data Abstraction & Virtualization
Semantic Interoperability
Data Access
Data Sources

Figure 54 – Common demonstration overview

7.2 DATA SOURCES

The first aim of the prototype is acquiring, processing, and registering data and metadata coming from the different data sources and digital infrastructures existing in the digital seaports. In this demonstration, two different data sources are integrated into the demonstration:

- PCS Traffic: Port Community System (PCS) is an electronic platform used for exchanging port and logistic data among the companies involved in the transport operation. It is provided by ValenciaPort. In the common demonstration, this data source provides two different types of data, the real time and historical vessel port call details[43] and the historical import and export trade data.
- AEAT customs: This data source offers aggregated statistics of the external trade provided by the Spanish Tax Agency (AEAT customs)[44].

The demonstration illustrates the mechanisms for common Access and management of these data sources integrated into the DataPorts Platform.

7.3 DATA ACCESS COMPONENT

An agent [2] is a piece of code focusing on the access to the available data source and performing the transformation into the common data model in order to allow a common understanding of the data. The Data Access component is basically a web-based tool to manage and run agents that integrates data into the DataPorts platform. The Data Access component offers different actions that can be performed through its User Interface. The main actions performed by this component are the following:

- Creating an agent
- Running an agent

[43] https://www.valenciaportpcs.net/portcalls/search/historic

[44] https://sede.agenciatributaria.gob.es/Sede/en_gb/estadisticas/estadisticas-comercio-exterior.html

- Management of the data sources.
- Management of the data sources connection information and logs.

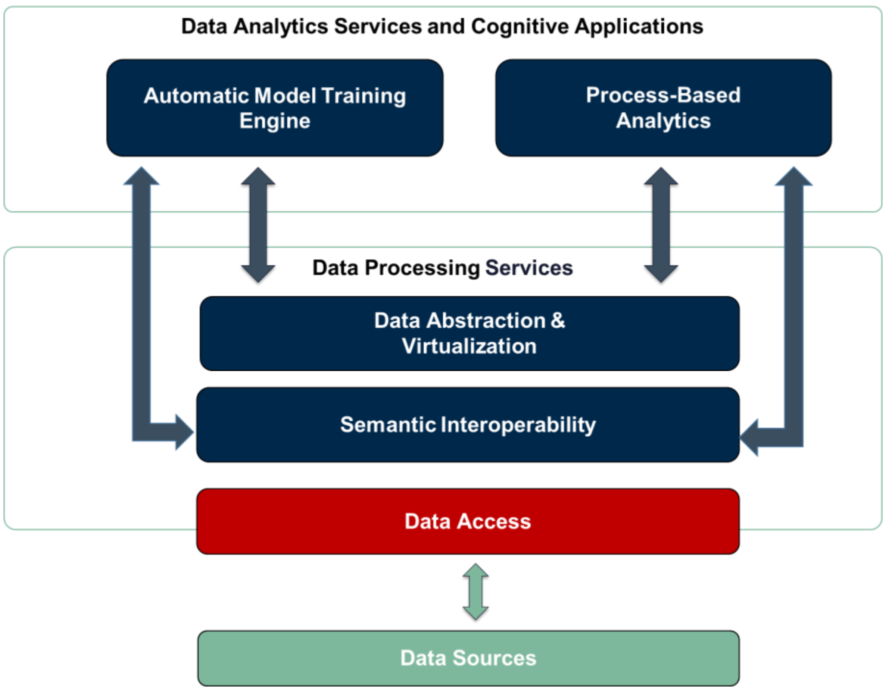

Data Analytics Services and Cognitive Applications
Automatic Model Training Engine
Process-Based Analytics
Data Processing Services
Data Abstraction & Virtualization
Semantic Interoperability
Data Access
Data Sources

Figure 55 – Position of the Data Access Component in the demonstration

Three agents have been developed and managed through the Data Access component to access the corresponding data sources. Two of these agents correspond to the PCS Traffic data source, one of them is publish/subscribe type and the other one is on-demand type. In addition, an on-demand agent has been developed for the AEAT customs data source.

As Figure 55 shows, the data sources will be finally consumed by the Automatic Model Training Engine. The agents involved in that information flow are provided in Table 1:

Data Source	Agent	Description of Agent role
PCS Traffic	On-demand	The agent accesses the PCS Traffic data source (for all the years indicated in the agent), performs the translation of the information into the common data model defined in the GIT repository and submits the information to the Data Abstraction and Virtualization Component. The information is sent divided into blocks of 100 entities and is reassembled by Fiware Cygnus.
AEAT customs	On-demand	The agent accesses the AEAT customs data source (for all the years indicated in the agent), performs the translation of the information into the common data model defined in the GIT repository and submits the information to the Data Abstraction and Virtualization Component. The information is sent divided into blocks of 100 entities and is reassembled by Fiware Cygnus.
PCS Traffic	Publish/Subscribe	The agent accesses the PCS Traffic data source (as planned when running the agent), performs the translation of the information into the common data model defined in the GIT repository and submits it to the Semantic Interoperability Component.

Table 1 – Data sources consumed by the Automatic Model Training Engine and their corresponding agents

In addition, Figure 55 also shows that the data Sources will be finally consumed by Process-Based Analytics. The agents involved in that information flow are the following (Table 2):

Data Source	Agent	Description of Agent role
PCS Traffic	On-demand	The agent accesses the PCS Traffic data source (for all the years indicated in the agent), performs the translation of the information into the common data model defined in the GIT repository and submits the information to the Data Abstraction and Virtualization Component. The information is sent divided into blocks of 100 entities and is reassembled by Fiware Cygnus.
PCS Traffic	Publish/Subscribe	The agent accesses the PCS Traffic data source (as planned when running the agent), performs the translation of the information into the common data model defined in the GIT repository and submits it to the Semantic Interoperability Component.

Table 2 – Data sources consumed by the Process-Based Analytics component and their corresponding agents

It is important to clarify that these two agents are the same as those referred to in Table 1.

7.4 SEMANTIC INTEROPERABILITY

The Semantic Interoperability component (SIC) provides a common API and data model to access the available data and metadata. The main functionalities offered by this component are:

- A common data model for the available data sources.
- Show the metadata of the available data sources and agents.
- Access to real-time and historical data available.
- Subscription to an available data source.

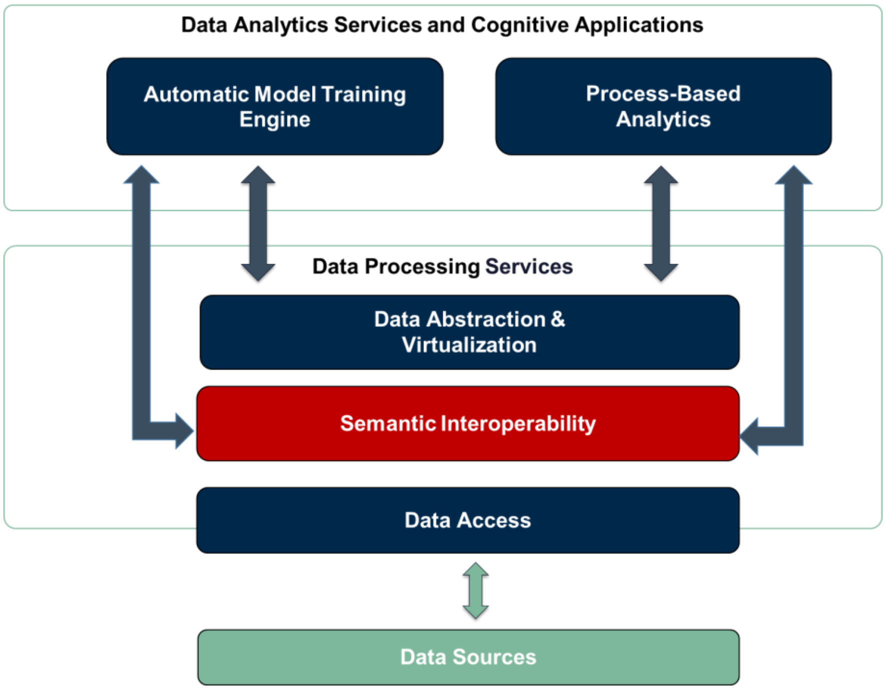

Data Analytics Services and Cognitive Applications
Automatic Model Training Engine
Process-Based Analytics
Data Processing Services
Data Abstraction & Virtualization
Semantic Interoperability
Data Access
Data Sources

Figure 56 – Position of the Semantic Interoperability Component in the demonstration

Following the information flow related to the data Sources that will be finally consumed by the Automatic Model Training Engine, the role of the Semantic Interoperability component is the following:

Data Source	Agent	Description of SIC role
PCS Traffic	On-demand	• DAV asks the SIC: An on-demand historical data request is sent to the On Demand subcomponent of SIC asking for the PCS Traffic historical information. This request in addition offers an endpoint where the on-demand PCS Traffic data will be received. • SIC asks the DAC: The On Demand subcomponent translates the query and asks the Data Access Manager to execute the PCS Traffic On-demand agent to provide the data expected. • DAC responds to the DAV: The PCS Traffic On-demand agent provides the data in the common format in the endpoint provided in the request.
AEAT customs	On-demand	• DAV asks the SIC: An on-demand historical data request is sent to the On Demand subcomponent of SIC asking for AEAT customs historical information. This request in addition offers an endpoint where the on-demand AEAT customs data will be received. • SIC asks the DAC: The On-Demand subcomponent translates the query and asks the Data Access Manager to execute the AEAT customs On-demand agent to provide the data expected. • DAC responds to the DAV: The AEAT customs On-demand agents provide the data in the common format in the endpoint provided in the request.
PCS Traffic	Publish/Subscribe	• AMTE subscribes to the PCS Traffic data source of SIC to receive incoming notifications. AMTE provides an endpoint to be notified. • AMTE receives a notification from the SIC in the endpoint provided when new PCS Traffic data is available and has been translated by the Publish/Subscribe PCS Traffic agent.

Table 3 – Role of the Semantic Interoperability Component in the access to the Data Sources consumed by the Automatic Model Training Engine

In addition, the data Sources that will be finally consumed by the Process-Based Analytics (PBA), the role of the Semantic Interoperability component is the following:

Data Source	Agent	Description of SIC role
PCS Traffic	On-demand	• DAV asks the SIC: An on-demand historical data request is sent to the On Demand subcomponent of SIC asking for the PCS Traffic historical information. This request in addition offers an endpoint where the on-demand PCS Traffic data will be received. • SIC asks the DAC: The On Demand subcomponent translates the query and asks the Data Access Manager to execute the PCS Traffic On-demand agent to provide the data expected. • DAC responds to the DAV: The PCS Traffic On-demand agent provides the data in the common format in the endpoint provided in the request.
PCS Traffic	Publish/Subscribe	• PBA subscribes to the PCS Traffic source of SIC to receive incoming notifications. PBA provides an endpoint to be notified. • PBA receives a notification from the SIC in the endpoint provided when new PCS Traffic data is available and has been translated by the Publish/Subscribe PCS Traffic agent.

Table 4 – Role of the Semantic Interoperability Component in the access to the Data Sources consumed by the Process-Based Analytics component

7.5 DATA ABSTRACTION AND VIRTUALIZATION

The Data Abstraction and Virtualization component (DAV) offers different processing and filtering techniques that are applied to the datasets obtained from a historical data request to the Semantic Interoperability Component. The aim of DAV is to provide processed and cleaned data ponds (by applying specific filtering rules) to the Data Analytics Services and Cognitive Applications layer.

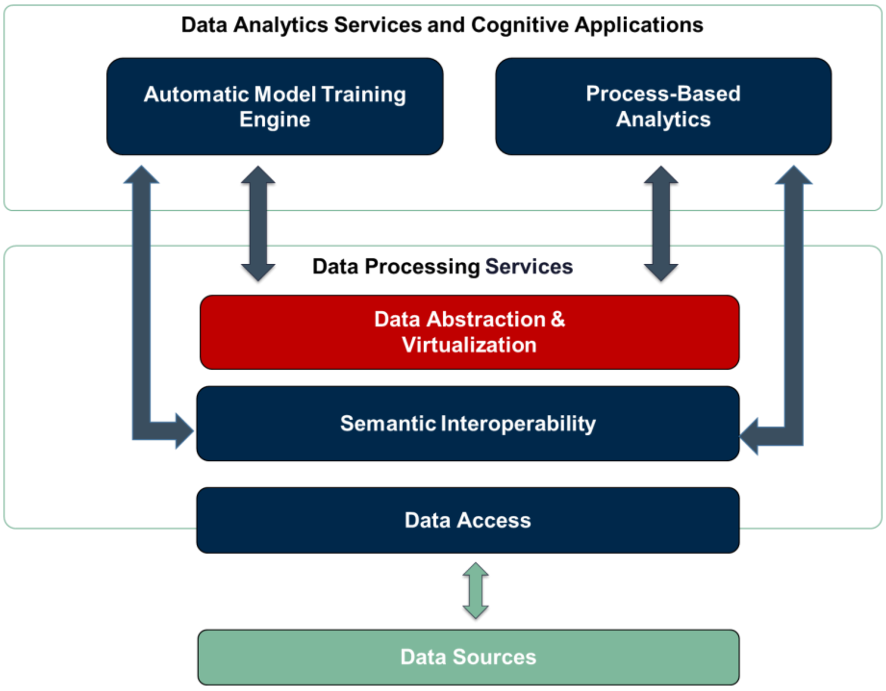

Data Analytics Services and Cognitive Applications
Automatic Model Training Engine
Process-Based Analytics
Data Processing Services
Data Abstraction & Virtualization
Semantic Interoperability
Data Access
Data Sources

Figure 57 – Position of the DAV component in the demonstration

More specifically, the steps followed by the component are as follows:

1. Data Analytics Services and Cognitive Applications modules ask for a specific dataset (id), in specific format (for example Parquet) and with specific filtering rules.
2. DAV checks if this dataset exists in the Virtual Data Repository (VDR), which is the DAV's temporary data lake.
3. If the dataset exists, then DAV applies those filtering rules to the existing dataset and forwards the result (data pond) to Data Analytics Services and Cognitive Applications layer in the requested format (for example Parquet)
4. If the dataset does not exist, DAV asks SIC's on-demand component for this specific dataset (id).
5. SIC's on-demand component fetches the data and notifies DAV. Then step #3 is executed.

To all the datasets obtained by the SIC's on-demand component, the following pre-processing techniques are applied by DAV:

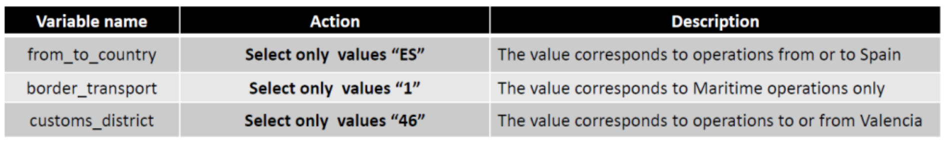
- Removal of whitespaces from all string-type cells
- Conversion of empty cells and 'NULL' string values to 'nan' values in all cells
- Removal of records / dataset rows without datetime values, or with wrong ones
- Conversion of datetime values to UTC format

Given that a dataset has been properly pre-processed and cleaned and is temporary stored in VDR, DAV can provide its metadata via a RESTful API.

The specific filtering rules applied to the (pre-processed and cleaned) datasets used in the demonstration are the following:

- PCS Traffic: rules for this dataset are depicted in Figure 43.

- AEAT:



Variable name	Action	Description
from_to_country	Select only values "ES"	The value corresponds to operations from or to Spain
border_transport	Select only values "1"	The value corresponds to Maritime operations only
customs_district	Select only values "46"	The value corresponds to operations to or from Valencia

Figure 58 – Rules for the AEAT customs dataset

7.6 AUTOMATIC MODEL TRAINING ENGINE

The Automatic Model Training Engine (AMTE) is a technical solution to create Cognitive Services for Ports' business KPIs. From a set of available datasets, this service provides a Training Web Interface (front-end) to automatically train the best machine learning model to answer a specific port KPI. Internally, the component implements a set of predefined training pipelines, using state-of-the-art ML algorithms for domains of regression, forecasting and classification, such as time series forecasting and values imputation. The front-end acts as the interaction bridge between a potential end-user and the model's training engine. The Web Interface also visually shows all the full potential of the component and summarizes the work done at all levels. The main utilities of the front-end are the following:

- Creating a new cognitive service
- Managing a trained cognitive service
- Exploring the machine learning models associated with a trained cognitive service
- Analysing and interacting with the outcome of a trained cognitive service

The historical data needed to train the machine learning models is downloaded by the Data Access component. The metadata will be provided by DAV to AMTE, so the information can be shown in the Web Interface. Then, the historical data will be properly pre-processed (structured, cleaned, formatted) and provided in parquet file by the DAV component. Finally, real-time data will be provided by the Semantic Interoperability component, also properly pre-processed, so the trained model can make new predictions.

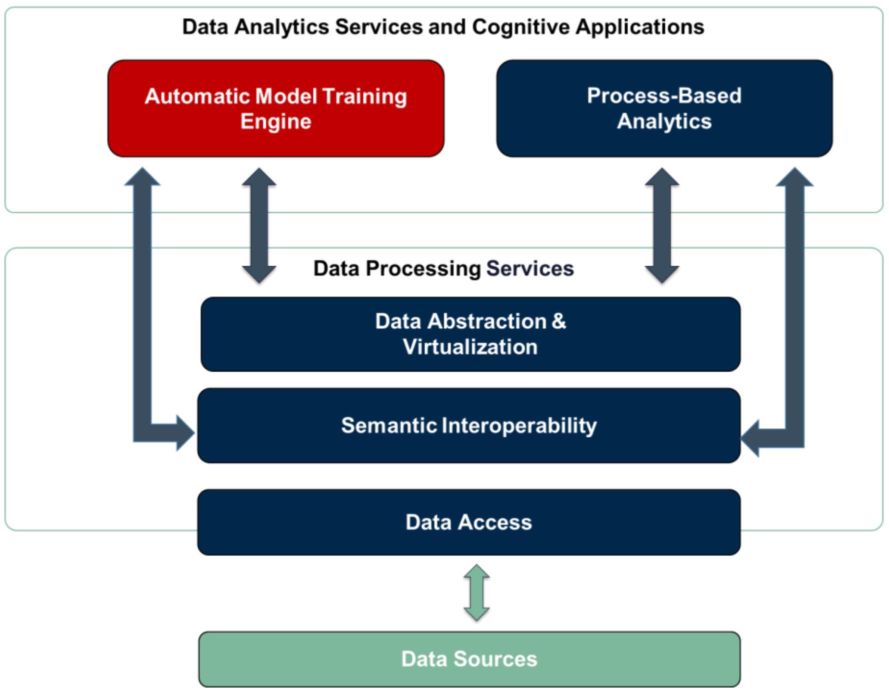

Data Analytics Services and Cognitive Applications
Automatic Model Training Engine
Process-Based Analytics
Data Processing Services
Data Abstraction & Virtualization
Semantic Interoperability
Data Access
Data Sources

Figure 59 – Position of the Automatic Model Training Engine in the demonstration

Regarding the data sources used in the common demo, the data processing services will offer a common way to access the input data to the Automatic Model Training Engine, so it can be utilized to create new Cognitive Services:

- Historical PCS Traffic datasets cleaned and processed: The specific cognitive service that would be trained is the one called "Vessel Time of Departure Estimator", which estimates the date and time of departure of an arriving vessel, based on the port/terminal/regular line associated to it. Once the arrival time is indicated, the ETD will be calculated.
- Notification of new PCS Traffic information via the publish/subscribe mechanism: Ideally, once trained with historical data, the service will be able to make new predictions over new incoming data coming from the Semantic Interoperability component. Historical AEAT datasets cleaned and processed: The Cognitive Service named "Customs Trade Volume" would be trained. This Cognitive Service performs an estimation of the volume of a certain type of good (in tons) from/to a specific district in a certain time horizon in months, selected by the end-user throughout the Training Web Interface. Hence, the expected trading forecast might be analysed by the user in order to make more accurate business decisions.

7.7 PROCESS-BASED ANALYTICS

The Process-Based Analytics component analyses business processes (BP) by using both historic and real-time data available inside the DataPorts platform. The goal of this component is to provide its predictive results to cognitive applications, which inform the end-users about the predictions. Internally, the Process-Based Analytics component uses the available datasets in the platform to train a state-of-the-art Machine Learning (ML) model that provides high-accuracy predictions about the outcomes of currently running business process, a Reinforcement learning agent that provides reliability score, and the explanations model that explains for the predictions of the ML model. Specifically, the ML model used in this component is an Ensemble of LSTM models, which, based on the use case and the BP needs, provides predictions in the form of regression or classification. The RL model used is a proximal policy optimization (PPO) algorithm that uses a neural network to directly represent a policy function. The explanation method used is Loreley, which generates counterfactual explanations to offer for explainability and actionability, i.e., the user is provided with an example of how to take action by changing the input given to the ML model in order to achieve the desired outcome.

The results of the Process-Based Analytics component, namely the prediction, the reliability score and the explanation will be shown to the end-user via a user interface. The user interface allows the end-user to interact with the component in the following way:

- Request and receive business process metadata
- Request and receive predictions and explanations for historical business process instances
- Subscribe to a specific business process or business process instance to receive real-time predictions and explanations when available.

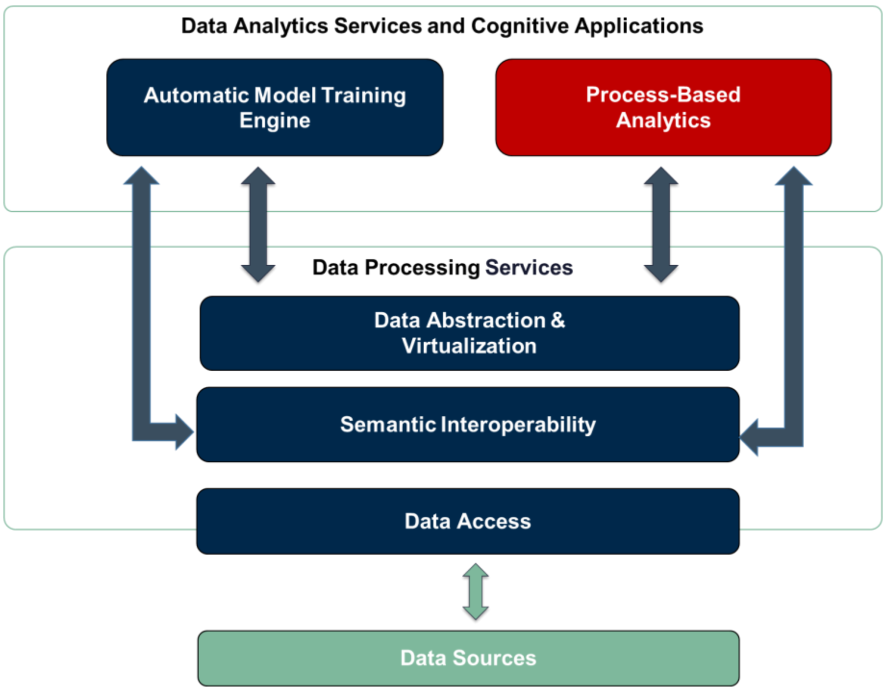

Data Analytics Services and Cognitive Applications
Automatic Model Training Engine
Process-Based Analytics
Data Processing Services
Data Abstraction & Virtualization
Semantic Interoperability
Data Access
Data Sources

Figure 60 – Position of the Process-Based Analytics component in the demonstration

The Process-Based Analytics component receives historical data from the Data Abstraction & Virtualization component and uses this data for training its sub-components. The historical data provided by the DAV component will be properly pre-processed (structured, cleaned, formatted) and provided in .csv file format.

The Process-Based Analytics component receives real-time data from the Semantic Interoperability component. The real-time data received should also be pre-processed and provided in the same format as specified for the historical data. Real-time data is then used by the Process-Based Analytics component to make real-time predictions.

Regarding the data sources used in the common demo, the data processing services offer Process-Based Analytics a common way to access the input data that will be analyzed by the models:

- Historical PCS Traffic datasets cleaned and processed: The specific cognitive service that would be trained is the outcome prediction service. This service uses historical and current information of the vessels in the port to predict if a vessel will leave the port in the estimated time or there will be a delay. Besides the prediction the service will also calculate a reliability score for the specific prediction. In the case of a predicted delay, an explanation will be generated to allow the end-user to understand the reason for the delay and how to proactively adapt.
- Notification of new PCS Traffic information via the publish/subscribe mechanism: Once trained with historical data, the outcome prediction service will be able to generate predictions and explanations about real-time running processes whose data will be provided by the Semantic Interoperability component. The outcome prediction model will provide the results in the same way as described for the historical data.

8 RELEASE SUMMARY

Table 5 shows the release information summary for the Semantic Interoperability Component and its subcomponents, including subcomponent name, source code and dependencies. Similarly, Table 6 shows the release information summary of the DAV component and its subcomponents.

Component	Subcomponent	Release Information	Details and Links
Semantic Interoperability Component	Orion Context Broker	Version	1.0.1
		Source Code	https://github.com/FIWARE/context.Orion-LD
		Documentation	https://fiware-academy.readthedocs.io/en/latest/core/orion-ld/index.html
		Docker Image	https://hub.docker.com/r/fiware/orion-ld/
		Dependencies	Mongo (service deployed with the docker-compose)
	On Demand module	Version	1.0.2
		Source Code	https://egitlab.iti.es/dataports/data_processing/ondemand
		Documentation	https://platform.dataports-project.eu/docs/semantic/
		Docker Image	-
		Dependencies	Orion Context Broker, Data Access Component (services deployed with docker-compose)
	Data Model	Version	1.02
		Source Code	https://egitlab.iti.es/dataports/data_processing/datamodel
		Documentation	https://egitlab.iti.es/dataports/data_processing/datamodel
		Docker Image	-
		Dependencies	-
	Demonstration tool	Version	1.0.2
		Source Code	https://egitlab.iti.es/dataports/data_processing/demonstration_tool
		Documentation	-
		Docker Image	-
		Dependencies	Orion Context Broker, On Demand Component

Table 5 – Summary table of Semantic Interoperability Component release information

Component	Subcomponent	Release Information	Details and Links
Data Abstraction and Virtualization Component	PaFS	Version	1.0.2
		Source Code	https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav/-/tree/main/PreProcessing%20%26%20Filtering%20Software%20(PaFS)
		Documentation	https://platform.dataports-project.eu/docs/dav/
		Docker Image	N/A
		Dependencies	Apache Spark cluster, MongoDB, On Demand Component, VDR
	VDR	Version	1.0.2
		Source Code	https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav/-/tree/main/Virtual%20Data%20Repository%20(VDR)
		Documentation	https://platform.dataports-project.eu/docs/dav/
		Docker Image	N/A
		Dependencies	Kubernetes, MongoDB
	VDC	Version	1.0.2
		Source Code	https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav/-/tree/main/Virtual%20Data%20Container%20(VDC)
		Documentation	https://platform.dataports-project.eu/docs/dav/
		Docker Image	N/A
		Dependencies	Apache Nifi, Apache Spark cluster, MongoDB, VDR

Table 6 – Summary table of Data Abstraction and Virtualization Component release information

9 CONCLUSIONS

This deliverable offers the final version of the Data Processing Services of the DataPorts platform. These building blocks offer the key functionalities to guarantee that the rest of the platform components have common, homogeneous, and processed access to the data sources available on the platform. The deliverable is mainly focused on the technical description and a complete release summary of each component of the Data Processing Services. To provide the software described in this deliverable, a code repository and a technical documentation repository have been deployed in the early stages of the project.

In more detail, this deliverable describes the implementation of the Data Processing Services of the DataPorts Platform, which are composed by the Semantic Interoperability Component and the Data Abstraction and Virtualization component. In addition, the improvements included in the new version of the Data Access component are also described in this document.

The work done on the Data Access component from M18 until M27 has been focused on the improvement of the management of the agents and the integration of the on-demand agents with the Data Abstraction and Virtualization component, as well as the provision of new agent templates to facilitate the integration of the platform with the pilots.

The Semantic Interoperability Component provides a common API to access the data from all the connected data sources, following a common syntax and data model. The complete version of this component is now available. This component provides the following benefits:

- Interoperability: Allows the connection of heterogeneous Data Sources. Offering a System of systems and Common Data Models.
- Ease of use and deploy: Makes the access to the available data and metadata simpler. The components can be deployed easily in a PC, Server, Cloud Infrastructure or in a Raspberry Pi.
- Modular, Scalable and Extensible solution adapted to the user needs: it could be extended with added value tools, and it is possible to scale the components on demand.
- Less development effort: Reduces the effort in creating added value applications relying in the data sources available in the platform.
- Fully compatible with the Fiware Ecosystem and common Open-Source Software: Joint adoption and contribution to standards. Use of common components and shared Open-Source code.

This component will be updated and extended in the following months to provide a complete integration with the Identity Manager and Blockchain, and to support Linked Data. Also, additional extensions might be included if new requirements from the pilots are identified.

Regarding the Data Model, a complete version has been defined, which will be used in the agents developed to integrate the platform in the different use cases (WP5) and will be updated until M30 following the feedback received from the pilots.

The Data Abstraction and Virtualization (DAV) component is responsible for the proper pre-processing, cleaning, and filtering of any incoming datasets, storing them to a secure and scalable data repository (data lake), and then making them available as data ponds to any potential recipients. Data ponds are created by applying specific filtering rules, defined by the users upon request. DAV also transforms and serves the data in a format selected by the user, such as Parquet or CSV. The three sub-components that perform the complete process are 1) the (Pre)Processing and Filtering Software, where the initial pre-processing, cleaning, and filtering of the incoming datasets takes place, 2) the Virtual Data Repository, as the main data lake of DAV, and 3) the Virtual Data Container, through which the distribution of specific data ponds to potential data recipients is achieved.

This component has been updated and extended to guarantee the integration with the Semantic Interoperability component and the Analytics components of the DataPorts Platform. This integration process has been described in the common demonstration of the Data Processing Services. The demonstration shows a prototype that makes use of real data sources to illustrate the process and it is aimed to verify its feasibility. Moreover, additional features might be implemented during the integration process towards the completion of the project's requirements.

Finally, it is crucial to highlight the next steps related to the Data Processing Services. These steps are mainly focused on achieving the complete integration of the Data Processing Services with all the components of the platform and deploying and evaluating the components in the corresponding WP5 pilots. In the period from the completion of this deliverable (M27) to the final delivery of the platform (M30), the focus will be on the following actions:

- Support from technical partners in the deployment and use of the data processing services in the WP5 pilots.
- Integration of the Data Processing Services with Data Governance and Security features according to the plans described in D3.7 [4] and WP4 deliverables.
- Test and validation of the use of components in the whole platform with all components integrated.
- Issue reporting.
- Maintenance of the components.
- M30 release of the DataPorts platform. The updates in the Data Processing Services will be reported in the code repositories and in the online technical documentation portal.

10 REFERENCES AND ACRONYMS

10.1 REFERENCES

[1] Watson, R. T., Lind, M., Delmeire, N., & Liesa, F. (2021). Shipping: a self-organising ecosystem. In Maritime informatics (pp. 13-32). Springer, Cham.

[2] DataPorts Consortium, “D3.1 Data access interfaces” June 2021.

[3] DataPorts Consortium, “D3.4 Permissioned Blockchain network M18” June 2021.

[4] DataPorts Consortium, “D3.7 Permissioned Blockchain network M27” March 2022.

[5] DataPorts Consortium, “D3.6 Data analytics services and cognitive applications M27” March 2022.

[6] DataPorts Consortium, “D2.4 Platform architecture and specifications” December 2021.

[7] DataPorts Consortium, “D2.1 Industrial Data Platforms and seaport community requirements and challenges” March 2021.

[8] DataPorts Consortium, “D2.2 Scalability, Interoperability and Definition Standards” December 2021.

[9] CIM, “GS CIM 009 - V1.5.1 - Context Information Management (CIM); NGSI-LD API,” 2021.

[10]“BDVA, FIWARE, GAIA-X and IDSA Launch an Alliance to Accelerate Business Transformation in the Data Economy.”, Online: https://design-principles-for-data-spaces.org/.

[11]“FIWARE for Data Spaces - FIWARE.”, Online: https://www.fiware.org/marketing-material/fiware-for-data-spaces/.

[12]“smart-data-models/data-models·GitHub.”, Online: https://github.com/smart-data-models/data-models/blob/master/specs/howto.md.

[13]“smart-data-models/data-models·GitHub.” Online: https://github.com/smart-data-models/data-models/blob/master/specs/howto_workgroup_to_code_a_datamodel.md.

[14]DataPorts Consortium, “D5.1 Integration, software quality assurance and deployment plan” March 2021.

[15]DataPorts Consortium, “D3.3 Data analytics services and cognitive applications” June 2021.

[16]DataPorts Consortium, “D3.2 Data processing services M18” June 2021.

[17]H. K. Gupta and R. Parveen, “Comparative Study of Big Data Frameworks,” IEEE Int. Conf. Issues Challenges Intell. Comput. Tech. ICICT 2019, Sep. 2019.

[18]O. C. Marcu, A. Costan, G. Antoniu, and M. S. Pérez-Hernández, “Spark versus flink: Understanding performance in big data analytics frameworks,” Proc. - IEEE Int. Conf. Clust. Comput. ICCC, pp. 433–442, Dec. 2016.

[19] J. Veiga, R. R. Exposito, X. C. Pardo, G. L. Taboada, and J. Tourifio, “Performance evaluation of big data frameworks for large-scale data analytics,” Proc. - 2016 IEEE Int. Conf. Big Data, Big Data 2016, pp. 424–431, 2016.

[20]D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, “A comparison on scalability for batch big data processing on Apache Spark and Apache Flink,” Big Data Anal. 2016 21, vol. 2, no. 1, pp. 1–11, Mar. 2017.

[21]C. Johnston, “Routing and Transformation,” Adv. Platf. Dev. with Kubernetes, pp. 337–378, 2020.

[22]I. M. A. Jawarneh et al., "Container Orchestration Engines: A Thorough Functional and Performance Comparison," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1-6, doi: 10.1109/ICC.2019.8762053.

[23]A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, “Persisting big-data: The NoSQL landscape,” Inf. Syst., vol. 63, pp. 1–23, Jan. 2017.

[24]A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, “NoSQL databases: Critical analysis and comparison,” 2017 Int. Conf. Comput. Commun. Technol. Smart Nation, IC3TSN 2017, vol. 2017-October, pp. 293–299, Feb. 2018.

[25]“NoSQL Key-Value Database Simplicity vs. Document Database Flexibility | NoSQL Key-Value Database Simplicity vs. Document Database Flexibility | InformIT.” [Online]. Available: https://www.informit.com/articles/article.aspx?p=2429466. [Accessed: 19-Apr-2022].

[26]D. Wu, S. Sakr, and L. Zhu, “Big data storage and data models,” Handb. Big Data Technol., pp. 3–29, Feb. 2017.

10.2 ACRONYMS

Acronym List	
AI	Artificial Intelligence
AMTE	Automatic Model Training Engine
API	Application Programming Interface
CEF	Connecting Europe Facility
CET	Central European Time
CSV	Comma-separated Values
DAC	Data Access Component
DAV	Data Abstraction and Virtualization
DoA	Description of Action
ETSI	European Telecommunications Standards Institute
IDS	International Data Spaces
IDSA	International Data Spaces Association
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
KPI	Key Performance Indicator
ML	Machine Learning
NaN	Not-a-Number / Missing Value
NGSI	Next Generation Service Interface
NGSI-LD	Next Generation Service Interface for Linked Data
OWL	Ontology Web Language
PaFS	Pre-processing & Filtering Software
PCS	Port Community System
PVCs	Persistent Volume Claims
REST	Representational State Transfer
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SIC	Semantic Interoperability Component
UI	User Interface
UML	Unified Modelling Language
URN	Uniform Resource Name
VDC	Virtual Data Container
VDR	Virtual Data Repository

WP	Work Package

Table 7 – Acronyms

11 ANNEX A: METADATA REGISTRY

The metadata registry data model defines the concepts for “DataSource”, “AgentImage” and “AgentContainer”.

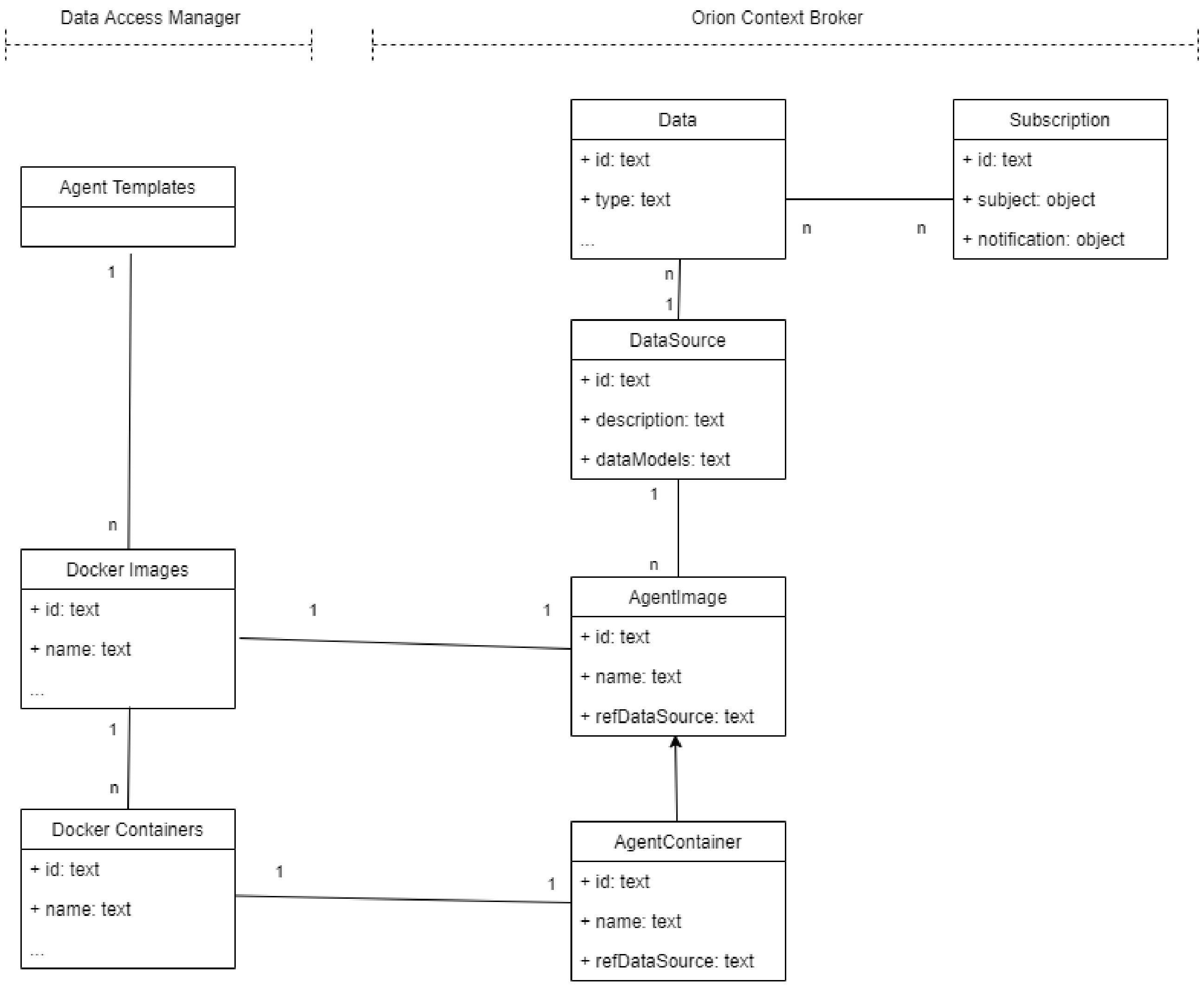


Figure 61 – Metadata registry data model

Each of the data sources connected to an instance of the DataPorts Platform is represented as a “DataSource” entity in the metadata registry. Data sources have the following attributes:

Attribute name	type	description	mandatory
id	text	Unique id of the data source (urn format)	Yes
type	text	Predefined value for the entity type that represents a data source (“DataSource”)	Yes
description	text	Human-readable description of the data source	No

dataProvided	object	Description of the entities provided by this data source. It represents the attributes provided by this data source. This object contains an attribute named “type”, which provides the value of the entity type of the data provided by the data source.	Yes
dataModels	text	Reference to the class of the Data Model (URL of the JSON schema)	Yes
service	text	Fiware Service value for the data of this data source (if different from the default value)	No
servicePath	text	Fiware ServicePath value for the data of this data source (if different from the default value)	No
onChain	boolean	Defines if the data of this data source is stored locally or on Blockchain. (Default: false).	No
attributes	object	JSON object describing the attributes defined in the data source that could be used in the requests to the On Demand component. These attributes are defined as parameters in the agent. Each attribute is defined as “name”:”data type”. If no parameters can be used for filtering, this object will be empty.	No
mapping	object	Defines the mapping between the attributes in the Data Model and the parameters of the agent listed in “attributes”. If no parameters can be used for filtering, this object will be empty.	No

Table 8 – Data source metadata

Regarding the agents, they are represented as two different types of entities. The first type is “AgentImage”, which represents the corresponding docker image for the agent. This type is mainly used for the management of the on-demand agents, since they are activated when a historical data request is received. An “AgentImage” entity has the following attributes (Table 9):

Attribute name	type	description	mandatory
id	text	Unique identifier of the agent image	Yes
type	text	Predefined value for the entity type that represents an agent image (“AgentImage”)	Yes

agentType	text	Type of NGSI Agent for this image (“on_demand”)	Yes
name	text	Name of the Docker image	Yes
accessURL	URL	URL for on demand data requests (endpoint of the Data Access Manager)	Yes
refDataSource	Relationship	Unique identifier of the corresponding data source	Yes

Table 9 – On-demand agent metadata

The other type of entity that has been defined for the agents is “AgentContainer”, which represents an agent that is running continuously or activates using a scheduler. This type is related to the publish/subscribe agents and has the following attributes (Table 10):

Attribute name	type	description	mandatory
id	text	Unique identifier of the agent container	Yes
type	text	Predefined value for the entity type that represents a running agent (“AgentContainer”)	Yes
agentType	text	Type of NGSI Agent for this image (“publish-subscribe”)	Yes
name	text	Name of the Docker image	Yes
refDataSource	Relationship	Unique identifier of the corresponding data source	Yes

Table 10 – Publish/subscribe agent metadata

12 ANNEX B: EXTERNAL EVALUATOR REVIEW

12.1 EXTERNAL EXPERT REPORT

Following recommendations after the first review meeting, the consortium has looked for external opinion about the choice of technologies used to implement the functionalities foreseen in the DataPorts platform, specifically the comment was:

"Another recommendation concerns the choice of the technology used to implement some modules. In this sense, the experts suggest the consortium to look for some professional advice (from technical architects for example) in order to validate how appropriate those technologies are in the context of a (big) data platform. For instance, one of the modules includes a software to filter and process incoming data. The technology chosen is a web application based on the Flask framework, which may not be suitable for handling big volumes of data."

The technical architect asked to carry out the evaluation has been Dr. Jordi Arjona Aroca. Dr. Arjona has a Telecommunications engineering degree by the Universitat Politècnica de València (UPV) in 2008. He has two MsC, one in industrial computer science, automation and robotics by the UPV and a second one in telematics engineering by the Universidad Carlos III de Madrid, where he also got his PhD in Telematic Engineering, cum laude, in 2015 in a joint program with IMDEA Networks Institute. During his PhD he had the opportunity to spend 6 months in Beijing, in the Institute for Computing Technology (ICT) of the Chinese Academy of Sciences and of 3 months in IBM India Research Labs.

Besides his PhD position at IMDEA Networks Institute, he worked for 2.5 years as a Postdoctoral Researcher, and later, as Member of Technical Staff, at Nokia Bell Labs in Dublin. Moving back to Spain, he worked at the Fundacion Valenciaport for 19 months, before joining ITI in April 2019. During these years he has participated in numerous European projects, even as a member of the External Advisory Board (RECAP). Similarly, he has multiple publications in international conferences and journals as well as some patents.

The following subsections collects his thoughts and findings after carefully reading the documentation available: D2.4 Platform architecture and specifications (M24), D3.5 Data Processing Services (M27) and D3.6 Data Analytics and Services and Cognitive Applications (M27). Each subsection collects feedback component by component, plus an overall platform analysis in the final conclusion's sections.

Finally, it is worth mentioning that the analysis has been carried out from a purely big data analysis perspective, not considering other project requirements. As an example, an IDS architecture implementation may imply to use a technology not suitable for big data analysis, it may happen that a technology which satisfies a concrete functionality may be a perfect fit for big data analysis, but incompatible with IDS architecture. In those cases, an analysis of pros and cons have been carried out together with a benchmarking exercise to find the most suitable solution for DataPorts. These cases are documented in WP3 deliverables.

12.1.1 Data Access Agents

Data ingestion in DataPorts is performed through the Data Access Agents. The selected tool is Cygnus, from the FIWARE ecosystem. Cygnus is based on Apache Blume and, hence, using a Source-Channel-Sink approach. Flume, or in this case Cygnus, is a solid solution for several reasons: Because, apparently, the complexity of the required data flows is low, only data ingestion is performed, and the work is on a Hadoop ecosystem. However, it could have been interesting to consider tools like Apache NiFi or, at least knowing why it has not been considered or why it has been discarded (for instance being an overkill). Considering the whole picture, e.g., the use of FIWARE components, Cygnus seems a solid and sufficient tool for this task.

12.1.2 Semantic Interoperability

The semantic interoperability component is necessary to ensure that data coming from the different ports involved in the project is homogeneous from an ontological point of view. Constructing this ontology is challenging because, even when data in different ports is or should be similar, the degree of digitization is very heterogeneous and so is the data available. Hence, the subsets of data available in each port may have had a small intersection. This however should be partially solved, or the impact smoothed, thanks to the participation of the ValenciaPort foundation, related to the port of Valencia, one of the most important ports in Europe, highly digitized and that has participated in projects devising partial ontologies on some of the most common procedures in ports. Assuming, then, that the partners have the knowledge and means to construct a solid data model, we must focus on the technologies used.

To this regard, the components used in DataPorts make sense from the point of view of their attachment to the FIWARE ecosystem. DataPorts uses the Orion Context Broker and NGSI specifications. These are widely known and accepted technologies that are, for instance, key components of the Connecting Europe Facility digital catalogue. They can be considered as solid components. Similarly, their use jointly with MongoDB seems solid enough. The main risk could have been that Orion cannot provide the functionality for requesting historical data, but this has been covered with a component devised and implemented in purpose. These solutions seem totally valid to, later, provide services or integrate with initiatives like IDSA brokers (to publish datasets metadata) or CKAN to publish the datasets themselves.

12.1.3 Data Abstraction and Virtualisation

The main technologies involved in the Data Abstracion and Virtualisation component in DataPorts are Apache Spark, Apache NiFi and MongoDB. The component is well structured and each layer well motivated and with clear goals defined. The selection of Spark for the Preprocessing and Filtering software (PaFS), focused on the preprocessing, cleaning, and filtering of data over other tools, such as Flink or Storm, is well justified from my humble point of view. Apache Spark is a well-known and widely used tool in the BigData community and I think its use is totally justified here, especially due to its scalability.

The functionality of the Virtual Data Repository (VDR) resembles a data lake. This data lake relies completely in MongoDB, which is, from my point of view, a perfectly valid and solid solution as it reflects its current positioning in the market. However, I am surprised that the solution only considers document-based storage. Although this is also a consequence of the previous layers, totally oriented to this type of data, it is relatively surprising that no other type of storage has been considered, like for storing objects. In any case, considering this limitation, the selection of MongoDB as storage technology is completely justified.

Finally, the selection of Apache NiFi and Apache Spark for the Virtual Data Container (VDR) seems to me, again, to be solid and completely justified, especially considering the ease for interacting with Spark from NiFi, or the usability of NiFi.

12.1.4 Process Based Analytics

The process-based analytics component is composed by three main subcomponents performing different types of monitoring: an ensemble predictive process, a prescriptive process, and an explainable predictive process. These seems to me three valid approaches to put in value the operation of the underlying components (data access, semantic interoperability, or DAV) moreover considering the kind of data available. However, I think the analysis we must perform is on the technologies used to do it. To this regard, the use of TensorFlow and Keras seems a perfectly valid approach, widely used in the community and in more complex and ambitious projects, that perfectly covers the need of this component. Same applies to the use of Python or Django.

12.1.5 Automatic Models Training Engine

There are several aspects to look at in this component. Starting with the training web interface, the use of frameworks like Angular or Django, even when I am not an expert in these matters, seems correct. In addition, we are talking about frameworks which are, again, widely used in similar circumstances and known to provide good results. The use of auxiliary tools like PostgreSQL is also justified and I do not think impacts in a substantial way and, finally, using NodeJS is quite a standard practice as well. Regarding the training distribution engine, the main technologies to use are Radiatus and Dask. Dask enables the parallelization of Python code and, given the availability of hardware resources, seems a valid option to easily leverage them without increasing the complexity on the design of the pipelines, which we will comment later. Radiatus, although not an extended tool given its early stage (TRL~7), will perfectly meet the purpose for what it is intended, also allowing a nice management of hardware resources. Coming back to the pipelines, using statsmodel, scikit learn and similar Python libraries is sufficient and solid, from my point of view, for the services being proposed.

Finally, we look at data formats and storing technologies. On the one hand, the use of Apache Parquet is justified, from my point of view, and brings in some advantages over standard row formats or json. Regarding the storage, I do not think using HDFS is totally justified over other alternatives like Minio. It is true, however that HDFS is sufficient and works smoothly with other tools used in this component, however, Minio may have been a better approach.

12.1.6 Final Conclusions

After reviewing the proposed architecture and the different elements in detail, I do not see or think that there is a technical risk. There can always be opinions on whether this or that piece of software could have been a better match for a particular functionality but, from my point of view, the selected technologies suffice to meet the goals of the project.

Moreover, I not only think that the technologies are sufficient considering the different blocks individually, in isolation, also I do not foresee that there may be problems regarding their integration to have a fully functional framework. Therefore, my feedback is positive, and I am not concerned about the future success of the project.

12.2 RESPONSE FROM THE AUTHORS OF THE DELIVERABLE

The authors of the deliverable have considered and appreciated the external expert report. It has offered a special value to improve the implementation of the platform and check the technical decisions from a purely Big Data analysis perspective. In any case, these technical decisions at the same time are considering the other requirements of the platform defined in WP2 of the project. For that reason, the main aim is offering the implementation of a data exchange and a data sharing platform capable to support the development and acceptance of both Big Data and AI applications.

As previously explained, this deliverable is focused on the Data Processing Services of the Platform and the updates of the Data Access component from M18. Therefore, this response is focused on the external expert report Sections 1.1, 1.2 and 1. 3.

Firstly, as a general comment the following sections of D3.5 are focused on the technical decisions evaluated in the report:

- Section 4.4: Explains the integration of the on-demand agents with the Data Abstraction and Virtualization component using Cygnus.
- Section 5.1: Explains the role of the Orion Context Broker in the Semantic Interoperability Component.
- Section 5.2: Explains the component developed to provide access to on-demand data.

- Section 5.5: Explains the technical decisions taken to provide a common data model for DataPorts.
- Section 6.2: Offers the conclusions of the comparative analysis carried out, concerning various Big Data frameworks, to implement the DAV component.

Secondly, for further clarification, two comments of the report need a more specific and explicit response. These comments are the following:

- *"However, it could have been interesting to consider tools like Apache NiFi or, at least knowing why it has not been considered or why it has been discarded (for instance being an overkill). Considering the whole picture, e.g., the use of FIWARE components, Cygnus seems a solid and sufficient tool for this task."*

  From the authors perspective persisting historical data is useful for Big Data analysis. Different components have been considered to perform this task. For example, tools like Apache Flume, Kafka, and NiFi offer great performance, can be scaled horizontally, and have a plug-in architecture where functionality can be extended through custom components. Mainly, the final decision was centred between the components offered by the Fiware ecosystem for data persistence. There are two of these components: (i) Cygnus, which uses Apache Flume technology, and (ii) Draco, which uses Apache Nifi technology cited by the external expert.

  The selection of the tool depends on the platform's needs. On the one hand, Cygnus is configured via configuration files. On the other hand, Draco offers a graphical interface to set up and monitor the procedure. In this situation there was no need of a configuration via a graphical interface; in fact, the developers indicated that this would have made the process of integrating the agents on-demand with DAV more complex. Therefore, the technology selected was Cygnus due to its simplicity.

  However, both technologies can be interchangeable. For that reason, if necessary, in the future it may also be considered to offer on-demand agent integration with Apache Nifi (Draco) to perform this task.

- *"However, I am surprised that the solution only considers document-based storage. Although this is also a consequence of the previous layers, totally oriented to this type of data, it is relatively surprising that no other type of storage has been considered, like for storing objects. In any case, considering this limitation, the selection of MongoDB as storage technology is completely justified."*

  From the authors perspective a document-based NoSQL database seems to be the optimum choice to implement the temporary data storage (VDR) inside the DataPorts Platform, according to the nature of the data that need to be handled. In fact, this type of database is ideal for semi-structured data without a fixed schema, but complying with certain formatting rules, such as XML, JSON, and BSON. In contrast, a relational database would require full knowledge of the incoming datasets structure beforehand, thus limiting the ability to store datasets with different schemas [23]. Since the data -in the context of DataPorts- are represented in JSON format, document-based storage should be selected, due to its high performance, flexibility, and scalability [24]. Compared to key-value NoSQL databases, document-based are preferable, to support different types of entities and enable complex querying, a feature that the platform should provide [25]. Considering opting for object-based storage, this architecture is generally more suitable for archiving large scale unstructured data such as photos, songs, etc., which is not the case in DataPorts [26].

Finally, even though this document corresponds to the final version of the implementation of the D3.5 components, in the coming months any technical change detected in the platform evaluation process or in the deployment of the use cases will be detailed in the online documentation of the platform and dissemination materials of the DataPorts project.