Title:	Document Version:
D3.2 Data processing services M18	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*:
M18 (June 2021)	M18 (June 2021)	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)

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, the present 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 plan for the future versions, 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	28/05/2021	First version of the document	Andreu Belsa (UPV), Matilde Julián (UPV) and Carlos E. Palau (UPV)
V0.2	07/06/2021	Version for internal review	Andreu Belsa (UPV), Matilde Julián (UPV), Anastasios Nikolakopoulos (ICCS) and Achilleas Marinakis (ICCS)
V0.3	15/06/2021	Update after review by UniKoblenz	Andreu Belsa (UPV), Matilde Julián (UPV), Anastasios Nikolakopoulos (ICCS) and Achilleas Marinakis (ICCS)
V0.4	25/06/2021	Update after review by Everis and ITI	Andreu Belsa (UPV), Matilde Julián (UPV), Anastasios Nikolakopoulos (ICCS) and Achilleas Marinakis (ICCS)
V1.0	25/06/2021	Version for Coordination review	Andreu Belsa (UPV), Matilde Julián (UPV), Anastasios Nikolakopoulos (ICCS) and Achilleas Marinakis (ICCS)
V1.1	29/06/2021	Final version	Santiago Cáceres (ITI)

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.

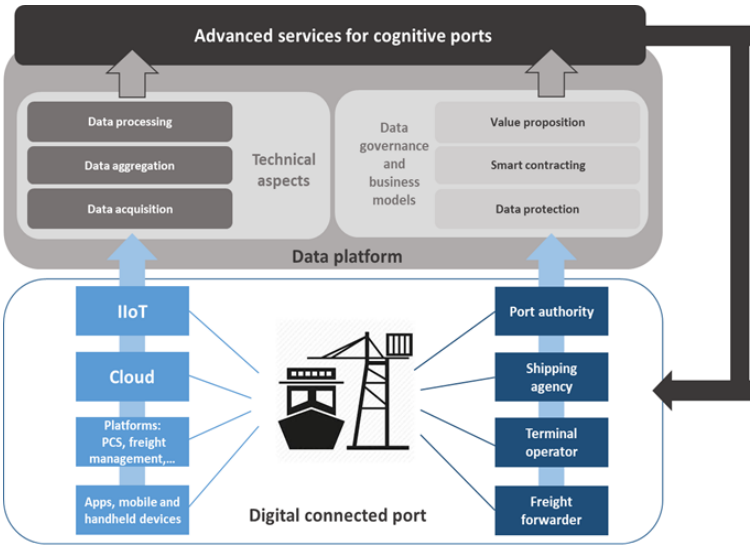


Figure 1 - DataPorts project overview

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.

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.
- D2.4 Platform architecture and specifications: Provides the architecture and specifications of the DataPorts Platform. An intermediate version of this deliverable is going to be delivered in M18.

Primary Dependant documents:

- 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 D3.2 to capture the complete picture of the joint work between these two components.
- 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 will present the status of this joint task from the technical point of view. However, the other perspectives of this joint work will be reflected in M24 in deliverable D2.2.
- D3.5 Data processing services M27: The final version of those services will be presented in deliverable D3.5 “Data processing services M27”.

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 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 5 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 6 Conclusions:** It contains the main 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. This M18 version of the document 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 will be presented in deliverable D3.5 Data processing services M27.

2 TECHNICAL OBJECTIVES

The following 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 is also composed by 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 D4.2 and D4.3, and with the "Analytics Services" described in D3.3.

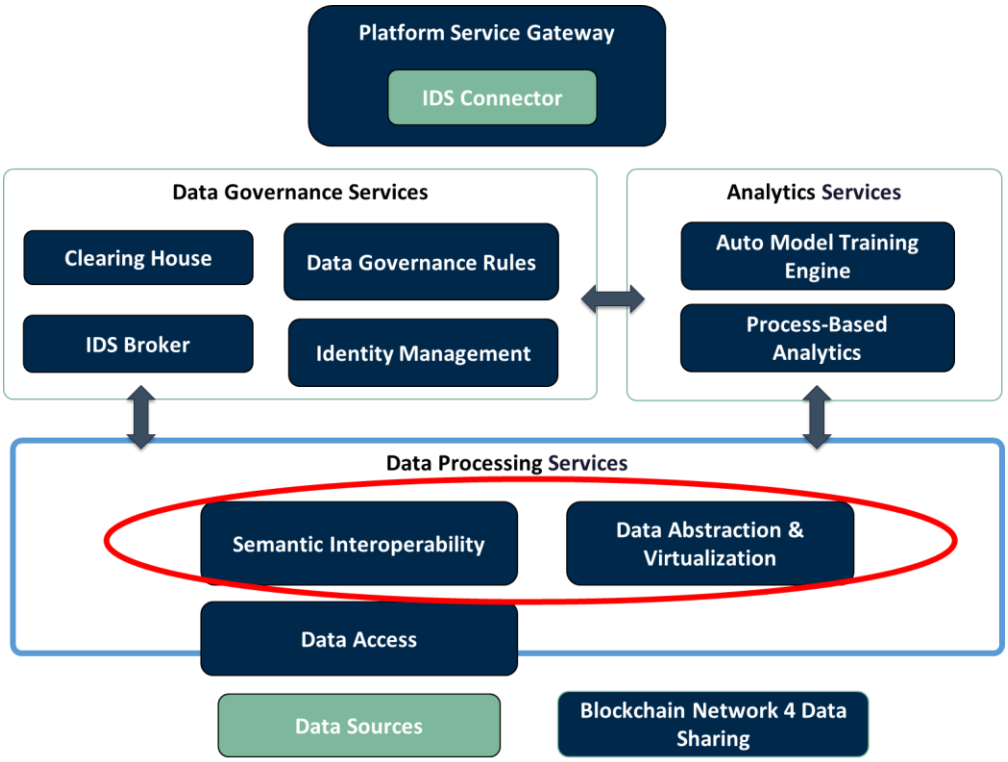


Figure 2 – DataPorts Platform building blocks

The position of the Semantic Interoperability Component in the DataPorts architecture, which is described in detail in D2.4 [3], 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 and Automatic Models Training Engine). 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 and the Automatic Models Training Engine. 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 repository and a description for the DataPorts Data Model and DataPorts Ontology. In addition, the Semantic Interoperability Component is connected to the Blockchain / Data Governance component through a Blockchain client and to the IDS Broker through an IDS connector.

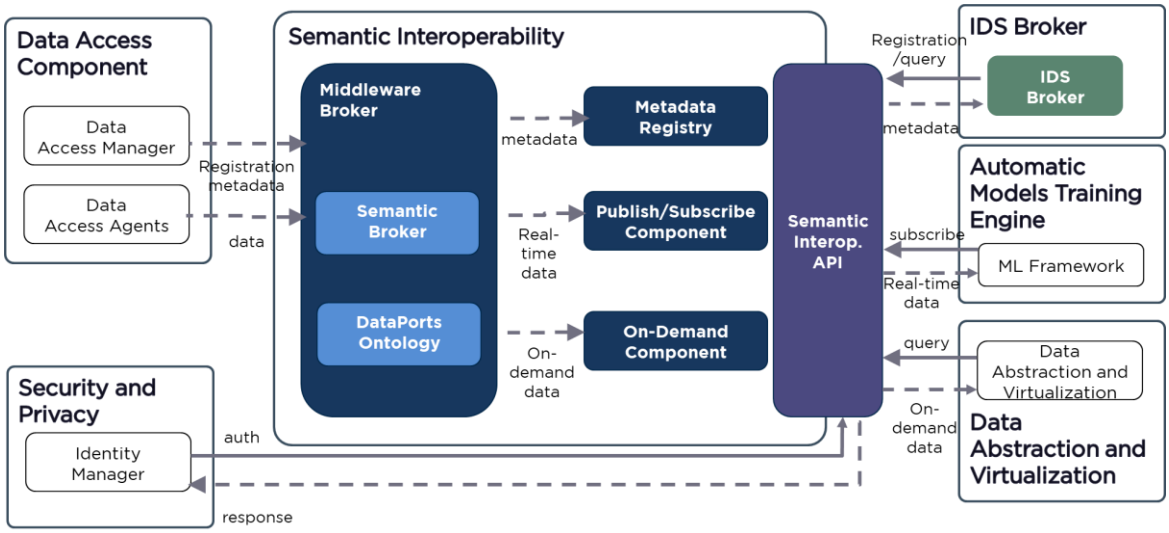


Figure 3 - Interactions of Semantic Interoperability Component

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 Automatic Models Training Engine and any other potential data recipient, through exposed RESTful APIs. 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 (mainly the Automatic Models Training Engine). The position of DAV in 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 5.

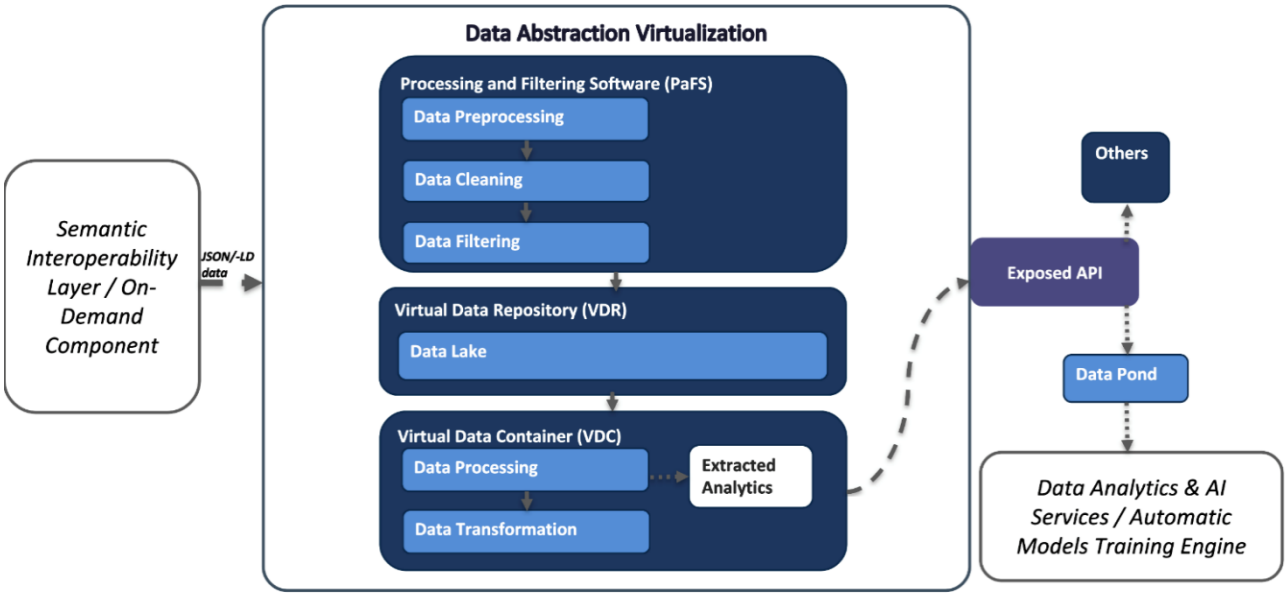


Figure 4 - Interactions of Data Abstraction and Virtualization component

4 SEMANTIC INTEROPERABILITY COMPONENT

This section covers the implementation status of the "Semantic Interoperability Component" in M18 of the project. This is the first document in which the component is described from a technical point of view; therefore, 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[1] and the Fiware Ecosystem[2], 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. The next and final version of this deliverable is expected in M27. The roadmap section covers the expected steps to achieve the technological objectives described in Section 2 and the expected contents for the M27 deliverable. Finally, Section 4.4 offers generic scenarios implemented until M18 to describe in a visual way the functionalities of this component and the potential interactions with other components of the platform.

4.1 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 IDS broker 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.

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 and D2.4, provide concrete details about those phases. However, an important part of the work during this period has been the definition of these relationships from a technical point of view, with the objective of ensuring that the pieces of the platform fit together during the implementation process.

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

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

This is mainly illustrated in a joint work with the Data Access component during these 18 months to provide both components fully integrated with each other. The components obtained from these two tasks have significant 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 and D3.2 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. 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 5 of this document) or the Data Analytics services and Cognitive applications (presented in deliverable D3.3).

Finally, a joint task with the WP2, WP3 and WP5 of the project is ongoing 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. Both the M18 and M27 versions of this deliverable will present the status of this joint task from the technical point of view. The other perspectives of this joint work will be reflected in M24 D2.2 “Scalability, Interoperability and Definition Standards” and in the results of the WP5 use cases.

4.2 TECHNOLOGICAL DESCRIPTION

The different blocks that comprise the implementation of the Semantic Interoperability Component are shown in Figure 5. The mappings between the subcomponents defined in the DataPorts architecture (Figure 3) and the technologies used in the 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) and Orion Context Broker.
    - **Metadata Registry**: The development is focused on an integration with the Data Access Manager (explained in D3.1), the Data Access Agents (explained in D3.1) 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).
- **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) and the Data Access Manager (explained in D3.1).
- **DataPorts Data Model**: Will be hosted in the project Git repository, with a specific organisation and guidelines. The development is 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 (Traefik[3]).

[3] https://traefik.io/traefik/

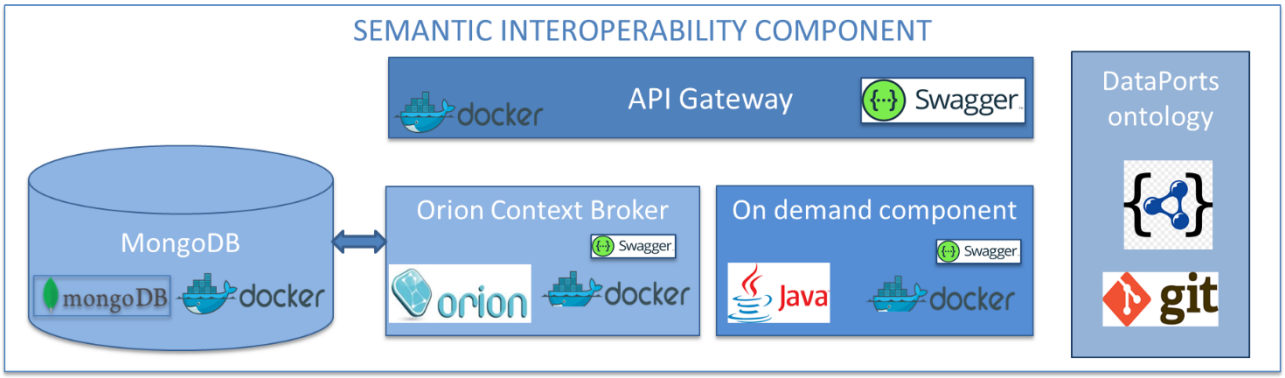


Figure 5 - Semantic Interoperability Component implementation

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).
- **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.

4.2.1 Orion Context Broker

As a result of the analysis of the requirements and the state-of-the-art presented in deliverable D2.1 [4], the Orion Context Broker[4], which also constitutes the main component in a Powered by Fiware[5] 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[6]. 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).

As Figure 6 shows, Orion manages the data flows from the agents to the components that act as data consumers and provides a NGSI[7] interface. The data managed by Orion is stored in MongoDB[8]. 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.

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

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

[6] CEF Digital Home (europa.eu)

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

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

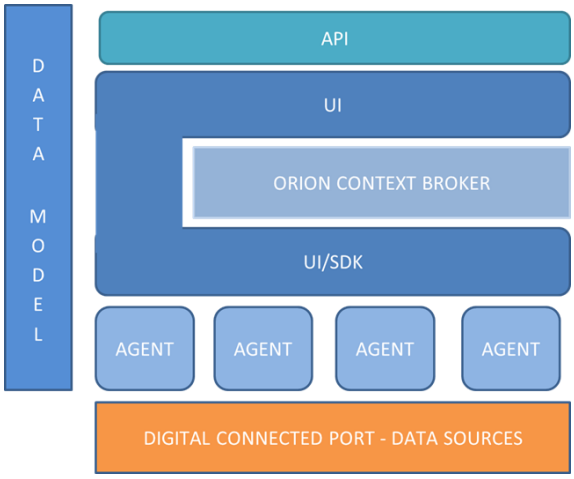


Figure 6 – Relationships between Orion Context Broker and the rest of the components

Regarding the Metadata Registry, the metadata is stored in a separate data base in MongoDB, named "metadata", which corresponds with the Fiware Service[9] 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. Figure 7 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.

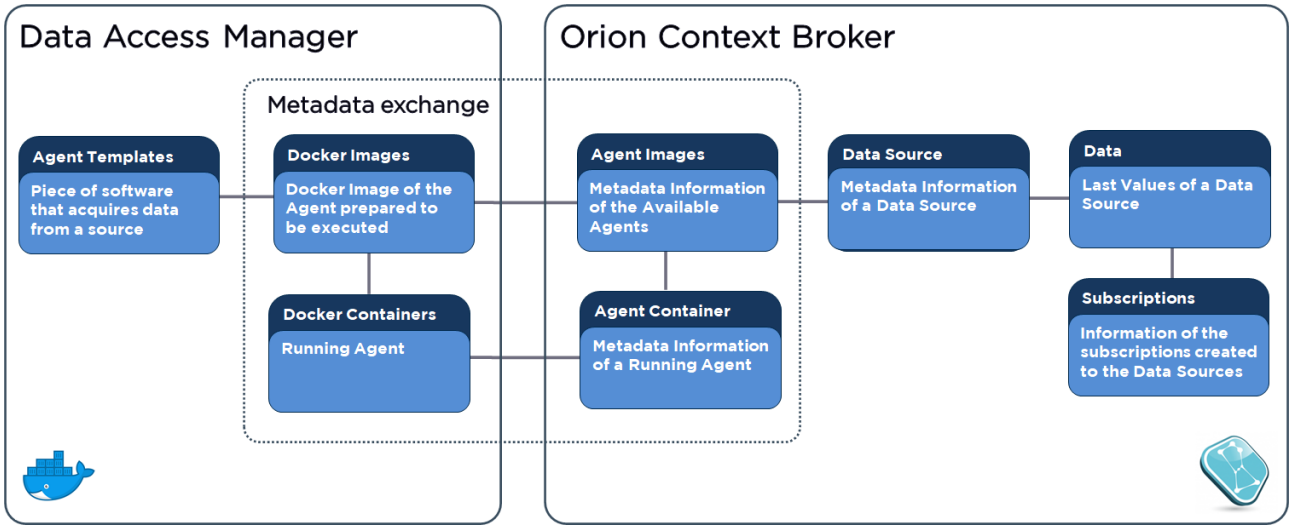


Figure 7 – Relationships between the data managed by Orion and the Data Access Manager

4.2.2 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

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

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. 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[10].

4.2.3 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 (Traefik[11]). 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 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).

4.2.4 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[12]. These components can be deployed together using docker-compose[13]. 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 Traefik are available on DockerHub[14]. 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, a docker-compose file for a joint deployment of these two components is also available.

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

[11] https://traefik.io/traefik/

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

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

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

4.2.5 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 to define the Data Model 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 [5]).
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.
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 will integrate concepts from existing ontologies and data models, including the Fiware Smart Data Models[15], the UN/CEFACT[16] data model, the SAREF[17] ontology and the IDSA Information Model[18]. The initial version of the Data Model will be hosted in a Git repository, which will contain the corresponding JSON schema documents describing the syntax in NGSI v2, as well as the documentation and examples. For each class in the Data Model, normalised (NGSI v2) and simplified (JSON) examples will be 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, future versions of the Data Model will be based on NGSI-LD[19], following the context information management standard defined by the ETSI (Section 4.2.1). 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. An OWL definition of the DataPorts ontology will be also provided.

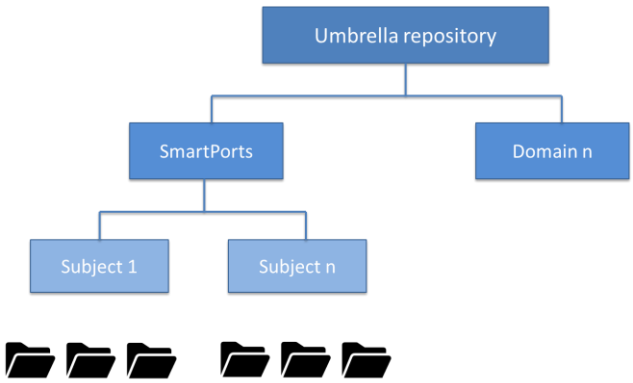


Figure 8 – Structure of the Data Model Git repository

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

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

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

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

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

The concepts defined in the Data Model will be organised by domain and subject, and this hierarchy will be reflected in the organisation of the Data Model repository. Thus, the Git repository[20] will have a tree structure (Figure 8), in which the root (Umbrella repository) will provide access to the repositories of the different domains and contain general information and guidelines, as well as common definitions and resources. Each domain will provide access to its different subjects and contain shared information and resources of the domain. Similarly, each subject will contain folders for each of its entity types, as well as shared information and resources of the subject. Finally, the specifications, examples, and information of each entity type will be provided in its corresponding folder. 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. 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).

4.2.6 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 9), as well as the metadata of the selected data source (Figure 10) 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 11), as well as the detailed information of the selected subscription (Figure 12). The Administration UI has been integrated with the Data Access Manager web UI (described in D3.1) in a new tab called “Orion”. Section 4.4 contains some examples of the use of this interface.

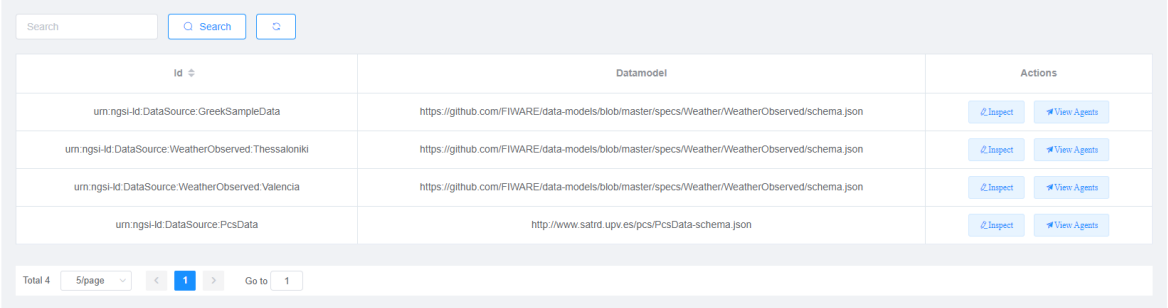


Figure 9 – Data sources metadata shown in the Admin UI

Figure 10 – Admin UI modal box of an example data source

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

[21] https://smartdatamodels.org/

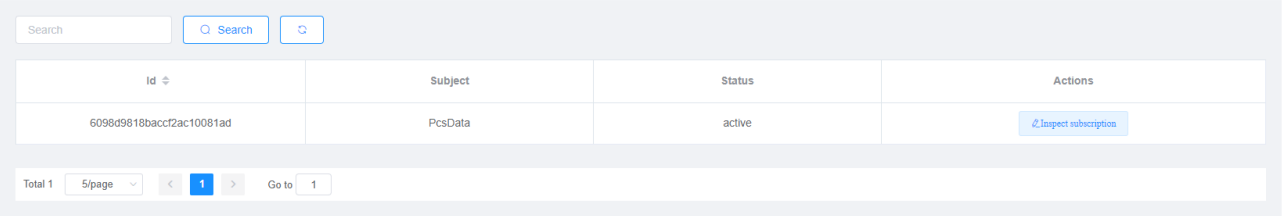


Figure 11 – Subscriptions in Orion Context Broker (shown in the Admin UI)

Figure 12 – Admin UI modal box of a subscription

4.2.7 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 4.2.6, 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 4.2.3 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 4.4 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 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.

4.3 DEVELOPMENT STATUS (M18)

As indicated in previous sections, development has focused on offering a first functional version of the component to avoid being a bottleneck for the other component's developments. It follows the requirements

and functionalities described in D2.1 and D2.4, respectively. The features listed in the following paragraphs have been prioritized to offer this first functional version of the component in the M18. Mainly, the work at this stage has been centred on the interaction with T3.1 because this task ends in the M18 of the project. The objective has been to offer a joint deployment that guarantees a functional Data Access Layer through an interoperable API. The other tasks carried out during this period have been focused on providing the basis for the next steps of the development. The final aim is to provide a data sharing and processing service fully integrated into the DataPorts Platform.

As a result of these efforts, a first functional version of the Semantic Interoperability Component has been developed. This version of the component is integrated with the Data Access Component and ready to be used by the WP5 Use Cases and the internal components of the platform. This initial version 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.
- Common interoperable API available providing the interfaces listed in Section 4.2.3.
- 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.

In addition, the following development tasks have been carried out during this period:

- Definition and development of the internal Data Model of the component.
- Definition of a methodology to perform the data modelling, as well as the guidelines for the specifications of the entity types and the repositories. The definition of the initial version of the DataPorts Data Model is still ongoing.
- Initial definition of the integration with IDS broker and Blockchain client using an API gateway.
- Implementation of a graphical testing tool to validate, demonstrate, and teach about the functionalities of the component.
- Integration of the component with external added value services like Cygnus[22], Node-RED[23], Apache NiFi[24], CKAN[25], Wirecloud[26], etc.

Currently, the Semantic Interoperability subcomponents are stored and managed in the following repositories:

- https://egitlab.iti.es/dataports/data_processing/semantic_interoperability_component_deployment

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

[23] https://nodered.org/

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

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

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

- https://egitlab.iti.es/dataports/data_processing/ondemand
- https://egitlab.iti.es/dataports/data_processing/datamodel
- https://egitlab.iti.es/dataports/data_processing/demonstration_tool

The code is managed inside the following subgroup:

- https://egitlab.iti.es/dataports/data_processing

The Administration User Interface is integrated in the Data Access Manager, for that reason, the code is available inside the following subgroup of the project:

- https://egitlab.iti.es/dataports/data_access

In addition, the source code of Orion Context Broker is available here:

- https://github.com/telefonicaid/fiware-orion

Regarding the plan for the future versions of the Semantic Interoperability Component until M27, the roadmap consists of the following expected tasks:

- 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.
- Improving the synergies between T3.2 and T3.3. The development efforts should be focused on refining the integration of the Semantic Interoperability Component with the Data Abstraction and Virtualization component. The aim is to offer these two components as the DataPorts Data Processing Services.
- Identifying existent open-source tools that could be integrated with the Semantic Interoperability component to solve a pending technical challenge or offer an added value to the component. For example, one of such tools might cover a semantic functionality identified or required in the GA, like providing mechanisms related with semantic data compression or offer data persistence connectors.
- Providing by M24 a harmonised version of the DataPorts common Data Model, which will be adopted by the project. This implies a joint work with WP2, WP3 and WP5 partners.
- Contributing to the Smart Data Models initiative with new Data Models.
- Defining NGSI-LD specifications of the DataPorts Data Model. DataPorts ontology.
- Moving from Orion Context Broker to Orion-LD as core element of the Semantic Interoperability Component. This also implies a migration from NGSI v2 to NGSI-LD. The motivation is that ETSI NGSI-LD is implemented as a standard mechanism for effective data exchange between different stakeholders.
- Providing REST API interaction with Linked Data. Including the semantic functionalities identified or required. Updating the Semantic Interoperability Component to enable NGSI-LD support; this could be done by adopting Orion-LD Context Broker[27] in the Semantic Interoperability component and adapting the other modules accordingly.
- Integration of the Semantic Component API in the DataPorts Platform common API.
- Complete integration of the Semantic Interoperability Component with Privacy, Security, IDS and Blockchain.
- Focus on the scalability, extensibility, and modularity of the component.
- Participation in the Data Spaces Synergy Group, which is an initiative promoted in the scope of the 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. The aim of the partners

[27] https://github.com/FIWARE/context.Orion-LD

involved in T3.2 is to find synergies between other projects for exchange knowledge and develop more effective data sharing services.

4.4 EXAMPLE OF USE: DEMONSTRATION

The following example of use is 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 in the DataPorts Platform.

4.4.1 Demonstration overview

Figure 13 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 4.2.7) 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.

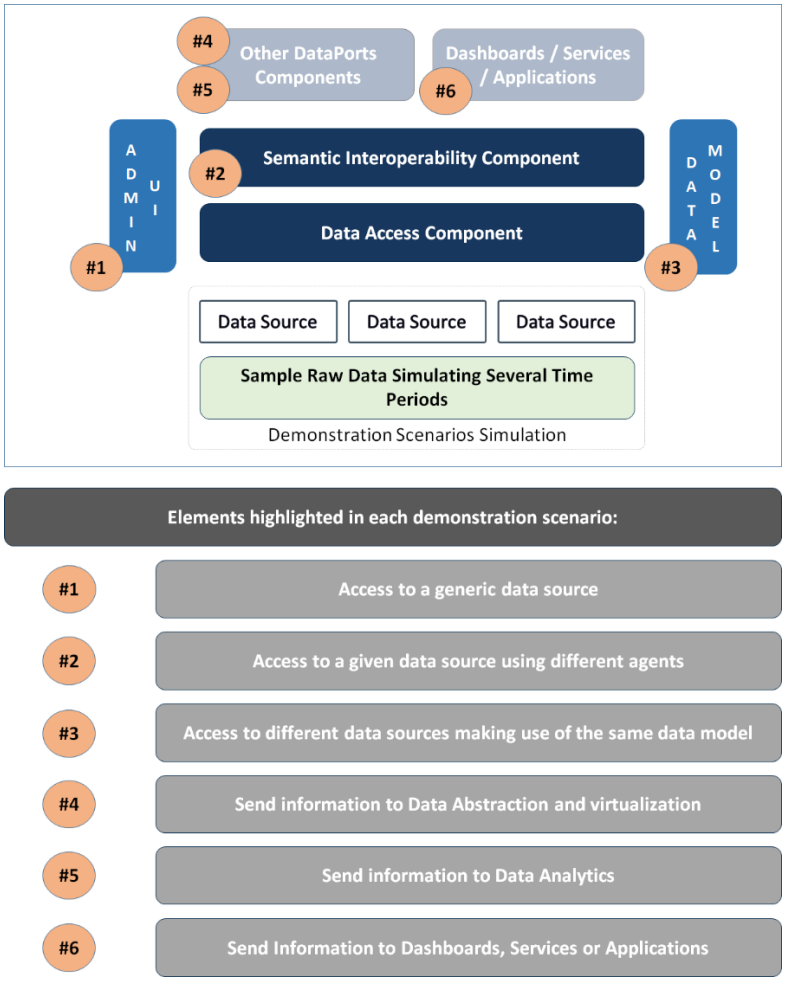


Figure 13 – Demonstration overview

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.

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

4.4.2 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) and uploaded to the platform. The different data sources (Figure 9) and agents (Figure 14), 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.

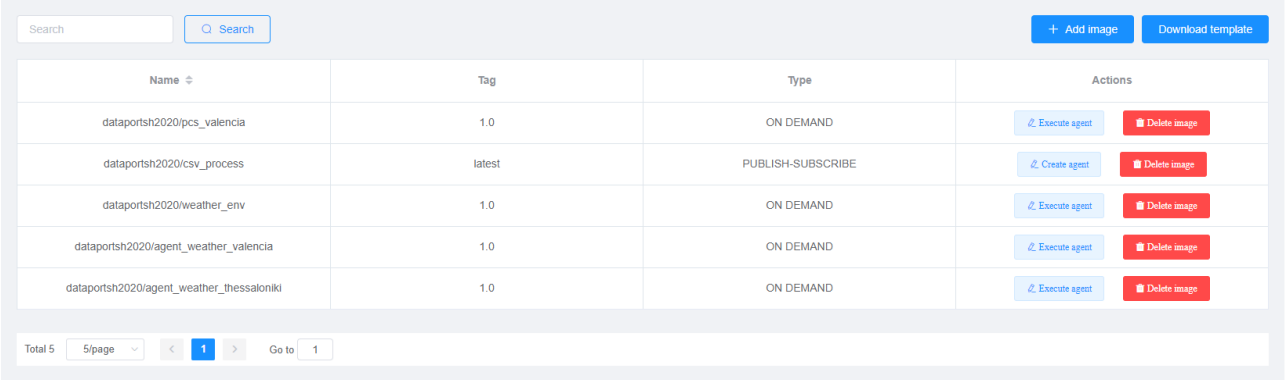


Figure 14 – Available agents (shown in the Data Access Manager UI)

4.4.3 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.

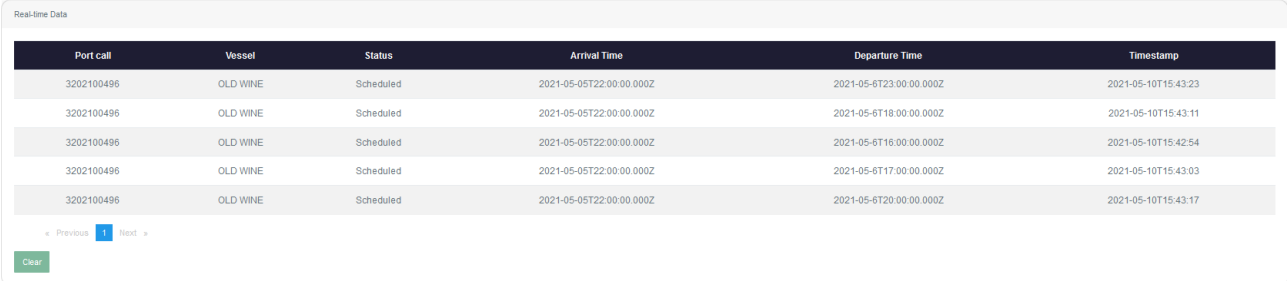


Figure 15 – Notifications (port call status changes) received in the demo application

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 demo application UI (Figure 15) and the details of the existing subscriptions can be shown in the Data Access Manager UI (Figure 11 and Figure 12).

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 16), which in this example represents the status of the port calls.

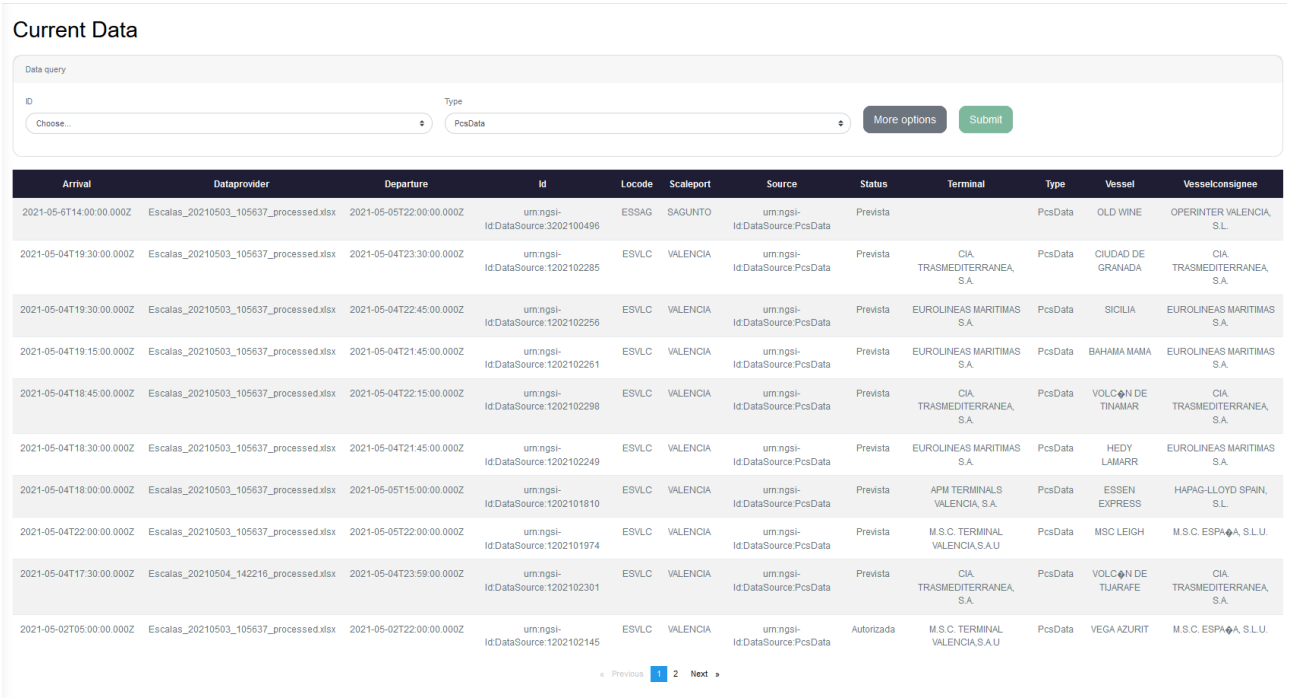


Figure 16 – Current PCS data (port calls) shown in the demo application

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 17).

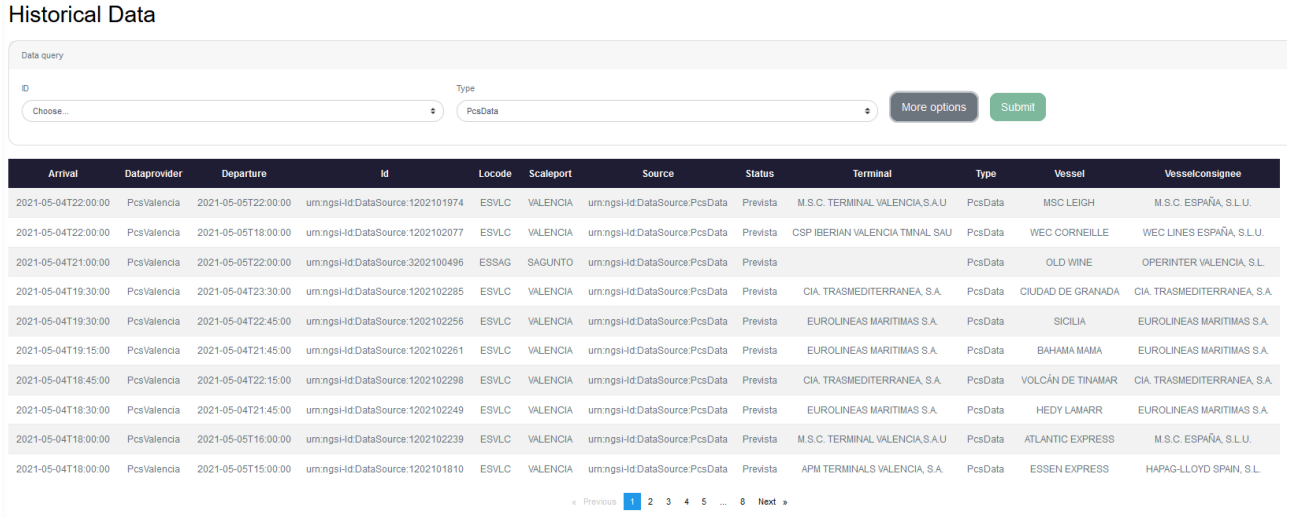


Figure 17 – Historical PCS data (port calls) shown in the demo application

In addition, the metadata of the data source (Figure 10) and agents (Figure 18 and Figure 19) can be shown in the Data Access Manager UI.

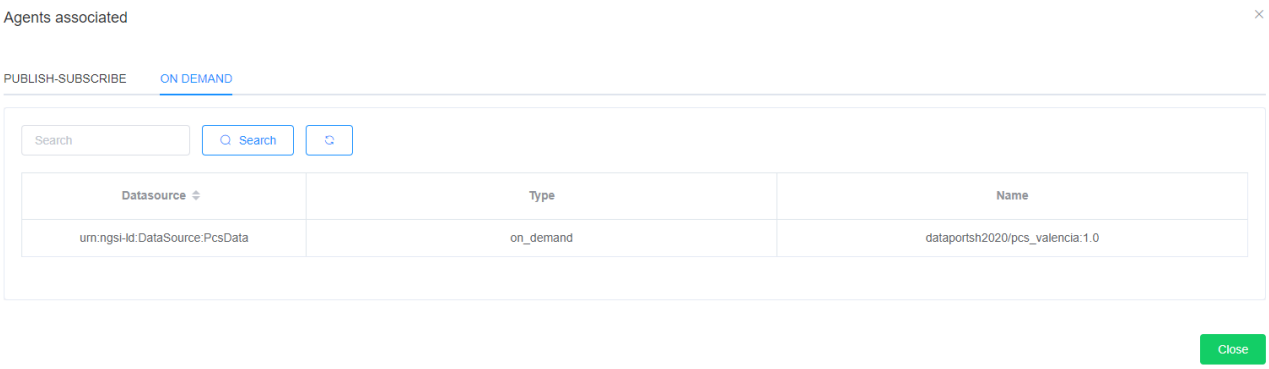


Figure 18 – On-demand agent registered for the example data source

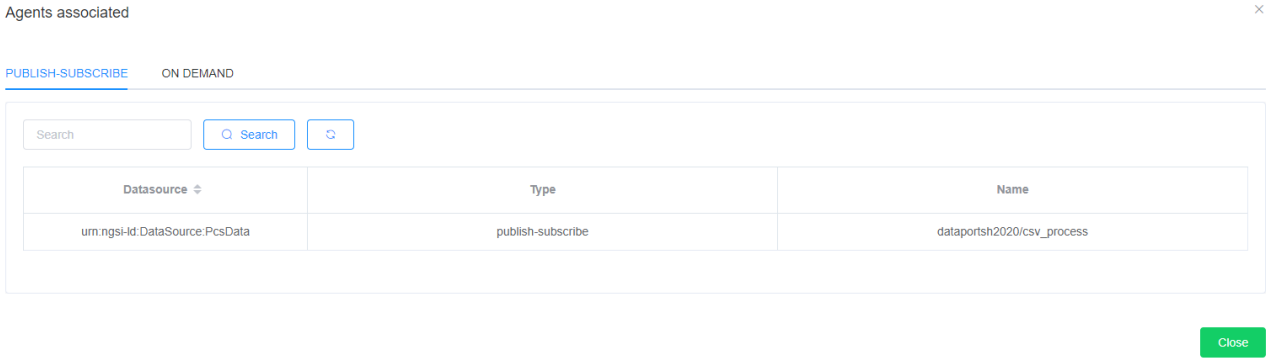


Figure 19 – Publish/subscribe agent registered for the example data source

4.4.4 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). 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 20, Figure 21 and Figure 22 show examples of the historical data obtained from the example data sources in this scenario.

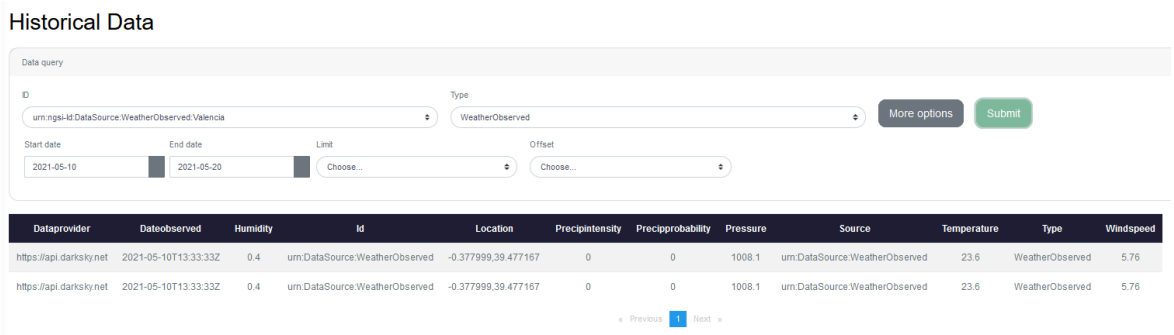
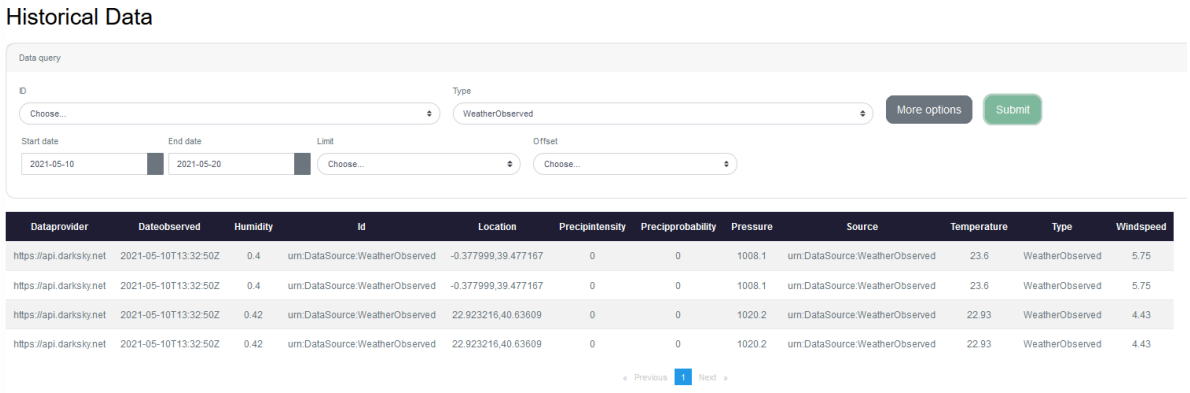


Figure 20 – Historical weather data obtained from the example data source of Valencia

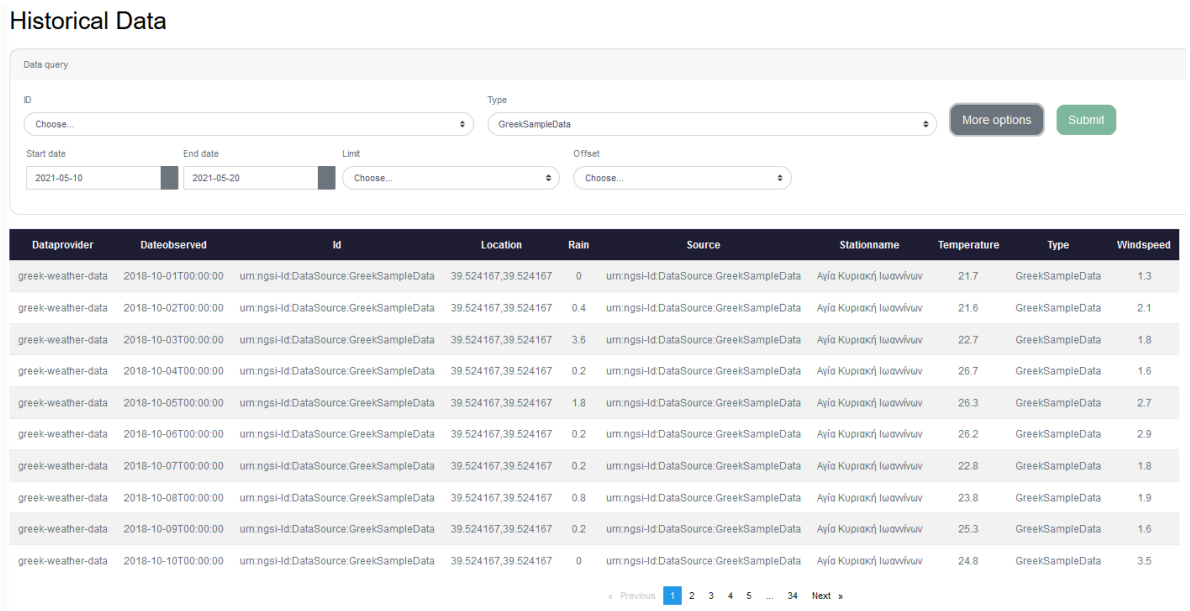
[29] https://github.com/smart-data-models/dataModel.Weather/tree/master/WeatherObserved



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

« Previous 1 Next »

Figure 21 – Combined historical weather data from two of the example data sources (Valencia and Thessaloniki)



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

« Previous 1 2 3 4 5 ... 34 Next »

Figure 22 – Historical weather data obtained from an historical database registry of several locations in Greece

4.4.5 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 4.2.6. Secondly, the DAV component can request the data to the On Demand component (Figure 23) 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 22 shows in a visual way the data which would be send to the DAV component. Then DAV performs its corresponding operations (Section 5) on the received data.

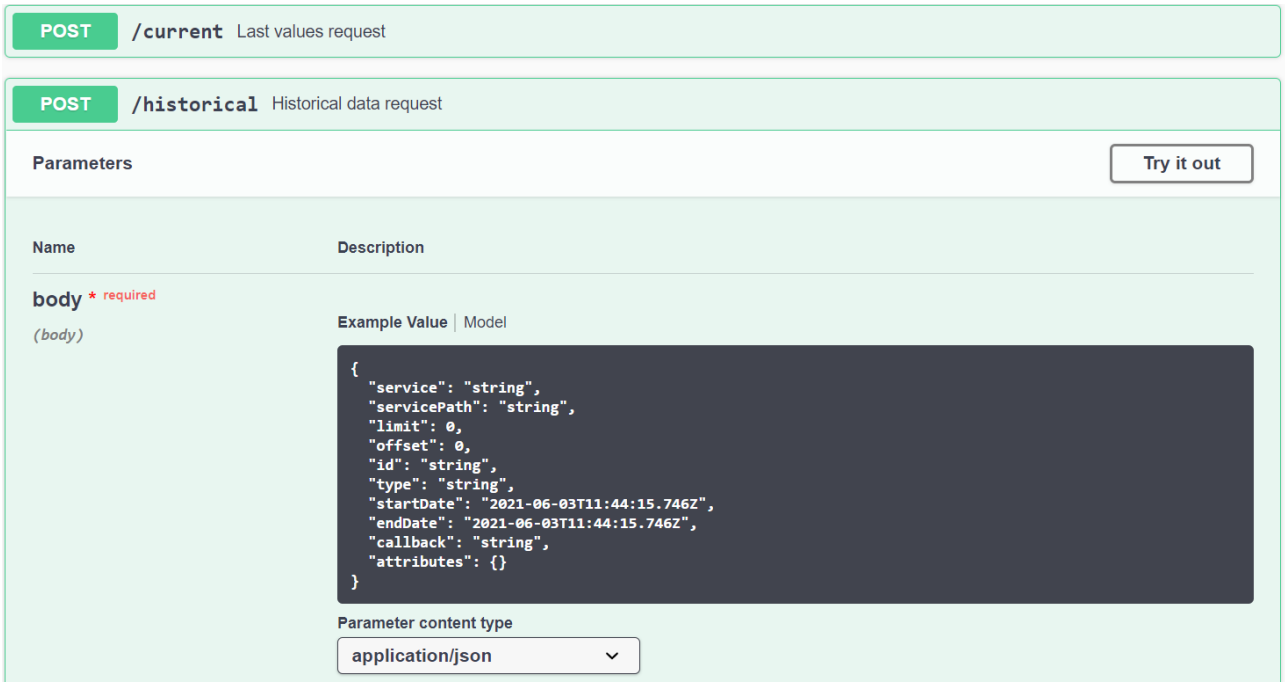


Figure 23 – On Demand component API

4.4.6 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 4.4.3), 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 [6].

4.4.7 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 4.2.3). 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 to 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 24 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/

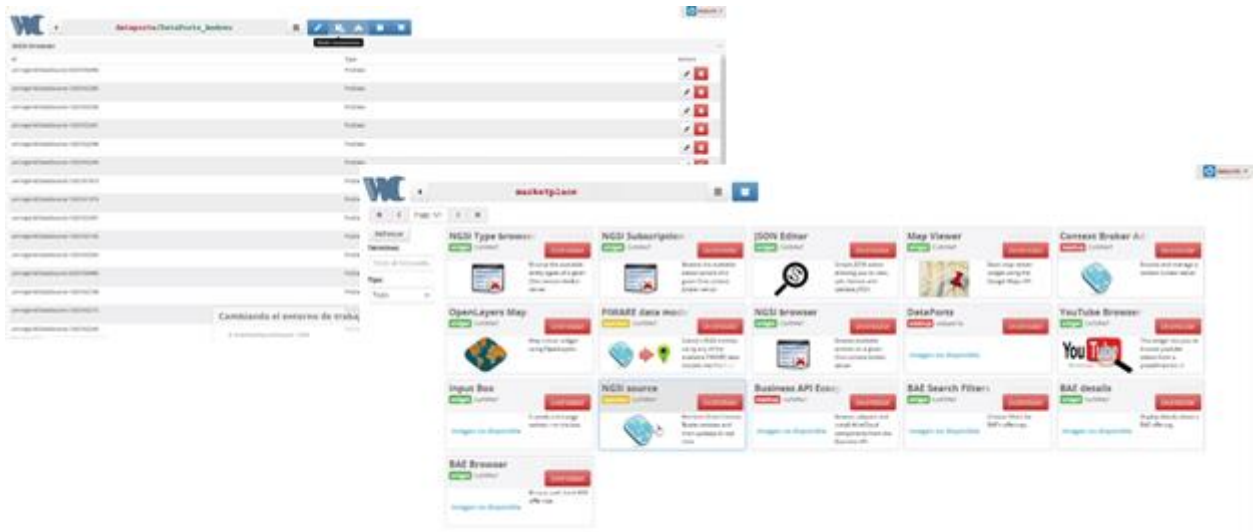


Figure 24 – Example of Wirecloud marketplace and a widget running using DataPorts data

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/

5 DATA ABSTRACTION AND VIRTUALIZATION COMPONENT

This section covers the implementation status of the "Data Abstraction & Virtualization" component in M18 of the project. This is the first document in which the component is described from a technical point of view; therefore, 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 sub-components of which Data Abstraction & Virtualization (DAV) is composed, their technological aspects and their development status. Furthermore, Section 5.3 presents the roadmap to fulfil the objectives described in Section 2, as well as the expected contents for the M27 deliverable, which is the next and final version of this document. Finally, Section presents a demonstration of DAV's usage and the sub-components' function.

5.1 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 exporting the "cleaned" & processed datasets through exposed RESTful APIs, thus making them available to any eventual client. Persistent data streams (that is, data that has already been collected and stored) will be the primary source of load for the DAV.

5.2 TECHNOLOGICAL DESCRIPTION

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 25, 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. The Virtual Data Repository 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. The Virtual Data Container is the layer between DAV and any potential data recipient. It may further process the data, in order to extract helpful analytics. Moreover, upon the dataset request by a recipient, it transforms the data into the desired format, whilst making them available to him/her.

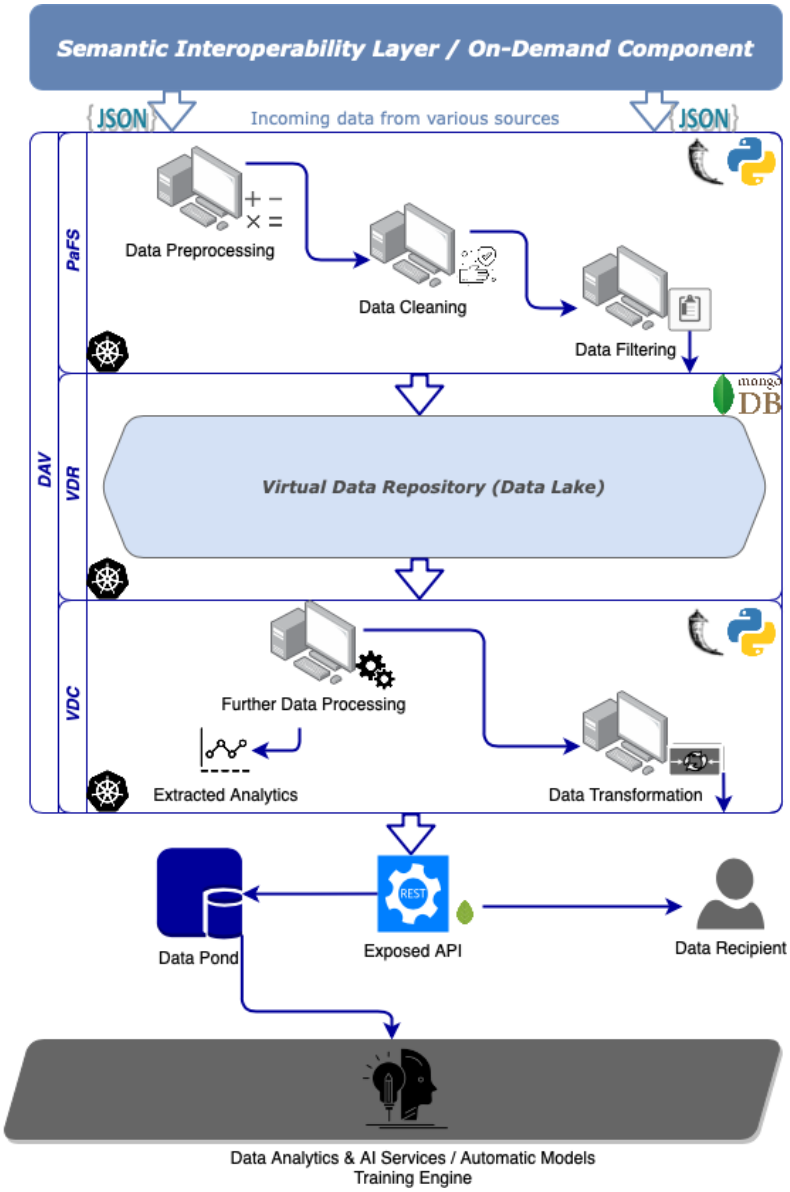


Figure 25 – Data Abstraction and Virtualization functionalities and technologies

As depicted in Figure 25, the whole DAV component lives inside a Kubernetes[35] 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 [7]. The version of Kubernetes upon which DAV is tested, is v1.20.5. The cluster consists of three physical machines (serves), 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. Inside this namespace is where all three sub-components will be functioning as pods & services.

5.2.1 (Pre)Processing & Filtering Software (PaFS)

Starting with PaFS, it is a sub-component that receives data streams through a GET REST API. 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

[35] https://kubernetes.io/

full dataset has been retrieved), PaFS creates a dataframe (a table), containing all the data. Then, the sub-component starts cleaning it from "dirty" values. Dirty values tend to be NaN, non-existing ones, zeros, outliers and wrong inputs (in areas / columns where these inputs are not acceptable e.g., outdoor temperatures at minus 100 degrees Celsius), which are most probably caused by sensors malfunction. PaFS detects these values and checks whether they are (or should be, based on each column) numerical or categorical ones. It then proceeds to replace the dirty values with means from their columns (for the numerical ones) and with the string "MISSING_VALUE" (for the categorical ones). It should be noted that, regarding the filtering (replacing) of the dirty numerical values with their columns' means, more details are taken under consideration. In case that the number of dirty values is too high, then the dataframe rows containing these values will be deleted, which means that no column mean replacing will take place. This way, the quality of data is assured. Replacing all numerical values with their columns' means, regardless of how many they are, would most probably affect the quality of any further processing of these data (e.g., applying machine learning techniques on it). Moreover, in case the number of dirty values is within the accepted limits, the way of means' calculation (e.g., linear or logarithmic) will be determined based on the column's type. To sum up, PaFS achieves the Preprocessing, Cleaning and Filtering of every incoming dataset. It preprocesses 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). A portion of PaFS' code can be seen in Figure 26:

```
def cleaner(dief):
    catcols = []
    numcols = []
    for c in dief.columns:
        #check if there are any strings in column
        if dief[c].map(type).eq(str).any():
            catcols.append(c)
        else:
            numcols.append(c)
    #create two DataFrames, one for each data type
    num = dief[numcols]
    cat = pd.DataFrame(dief[catcols])
    # create a num imputer for numerical valuees
    numimputer = SimpleImputer(missing_values=np.nan, strategy='mean')
    # we apply the numimputer only to numeric columns / values
    num = pd.DataFrame(numimputer.fit_transform(num), columns = num.columns)
    # create a cat imputer for categorical valuees
    catimputer = SimpleImputer(strategy='constant', fill_value="MISSING_VALUE")
    # we apply the catimputer only to categorical columns / values
    cat = pd.DataFrame(catimputer.fit_transform(cat), columns = cat.columns)
    #join the two dataframes back together
    newdief = pd.concat([cat, num], axis = 1)
    # return the cleaned dataset
    return newdief
```

Figure 26 – PaFS code part of initial filtering & replacing of the dirty values

Before finishing, PaFS proceeds to draw a correlation matrix between the columns of each dataset. This new piece of data will be stored along with the existing (cleaned) dataset at the Virtual Data Repository (analysed

later on). PaFS main code is written in Python[36] programming language and, more specifically, is a Python Flask[37] module. However, for the code to run inside the Kubernetes cluster, some steps must be completed. The steps are as follows:

1. Make sure that Kubernetes points at the namespace created specifically for DAV
2. Create a Dockerfile for the PaFS Python Flask script and build it with name "pafs:latest" (Dockerfile will search for the existence of an additional file named "requirements.txt")
3. Check that the image has been properly created and running
4. Create a .yaml file named "deployment.yaml" specifying all the necessary details for the PaFS docker image to function properly inside the Kubernetes framework (such as the NodePort information, for PaFS to have a GET REST API available in order to obtain incoming datasets)
5. Apply the deployment.yaml file to the Kubernetes framework
6. Check that PaFS pod & service have been properly launched

5.2.2 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 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[38], 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[39], as already mentioned in the deliverable 2.1. 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. 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:

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

[37] https://flask.palletsprojects.com/en/2.0.x/

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

[39] 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 final 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.

5.2.3 Virtual Data Container (VDC)

Moving on to the third and final sub-component of DAV, 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. Similar to PaFS, VDC is written in Python, as a Python Flask module. Moreover, it is a generic (sub) component, meaning that it can be slightly modified to match different kinds of needs. An instance of this generic software tool, "listens" to REST requests from potential data recipients, communicates with VDR to retrieve the queried data pond, it collects, prepares and finally converts it to a Parquet[40] format, where it gets forwarded to the data recipient as a Parquet file. Therefore, the VDC provides RESTful APIs to the end user, for communication with VDR. The end user makes a request to VDC, which then communicates with VDR, since the data lake is an internal component (living and communicating strictly within the Kubernetes cluster). It then queries MongoDB and gets data-response. Following that, it creates a new dataframe from the response, which is then converted into a table ready for Parquet transformation. Lastly, it saves the table in a .parquet file, which sends to the user who made the request. It is very important to mention that VDC is capable of hosting pre-defined methods, which will process the data-responses from VDR and perform more specific tasks (for example, calculating the standard deviation of daily temperatures during August 2009, in a weather dataset), in order to return aggregated data in Parquet format. An example of MongoDB query (part of the Python Flask module), regarding this specific VDC instance, can be seen in Figure 27. It describes a VDC function to query (after receiving a request at the route defined) a specific weather dataset and get all the documents (table rows) that match a user given year value. It gathers the queried response, prepares it and converts it to Parquet format. The created Parquet file is then forwarded to the data recipient / user.

```
@app.route('/tempOver') # it reads the parameter given in the 'temp' value, after ?.
#Example request to work http://127.0.0.1:5000/tempOver?temp=15
def tempOverer(name=None):
    # get the temperature parameter given
    temp = request.args.get('temp')
    # initialize a list where the mongo query response will be stored
    stringer = []
    # connect to mongo
    myclient = pymongo.MongoClient("mongodb://mongo-0.mongo:27017,mongo-1.mongo:27017")
    mydb = myclient["vdrDb"]
    mycoll = mydb["weatherdata"]
    # query mongo and save response to the list
    for x in mycoll.find({"MEAN_TEMP": { "$gt": float(temp) }},{"_id": 0}):
        stringer.append(x)
    # save response as a dataframe
    framed = pd.DataFrame(stringer)
    # convert dataframe to table ready for parquet format
    table = pa.Table.from_pandas(framed)
    # save the table as a parquet file
    pq.write_table(table, f'files/temp.parquet')
    # send the parequet file created as an attachment
    return send_file('files/temp.parquet', as_attachment=True)
```

Figure 27 – VDC method example

For VDC (as a Python Flask module) to run within the Kubernetes framework / cluster, some steps must first take place. These steps are very similar (if not quite the same) with the ones in PaFS case, and are as follows:

[40] https://parquet.apache.org

1. Make sure that Kubernetes points at Data Abstraction & Virtualization's namespace
2. Create a Dockerfile for the VDC Python Flask script and build it with name "vdc:latest" (Dockerfile will search for the existence of an additional file, tailored to VDC's python file, named "requirements.txt")
3. Check that the image has been properly created and running
4. Create a .yaml file named "deployment.yaml" specifying all the necessary details for the VDC docker image to function properly inside the Kubernetes framework (such as the NodePort information, for VDC to receive REST requests from external – outside the cluster – data recipients)
5. Apply the deployment.yaml file to the Kubernetes framework
6. Check that the VDC pod & service have been properly launched

5.3 DEVELOPMENT STATUS (M18)

During the first reporting period of DataPorts, the main goal was to develop an initial version of the Data Abstraction and Virtualization functional component, which would be the basis for the next steps of implementation. Special emphasis was put on identifying potential dependencies, constraints, and limitations related to DAV's communication with the other functional components of the DataPorts Platform, and particularly concerning the interaction of DAV with the Semantic Interoperability component and the Analytics components, in terms of data format, communication protocols, and APIs. Finally, DAV is designed and developed to cover certain requirements and functionalities of the project, as these are described in D2.1 and D2.4 respectively.

In this subsection, the features (sorted by subcomponent) of DAV that are already included in this first functional version are listed, as well as additional functionalities that are planned to be implemented, until the end of T3.3 and WP3 in M27.

- Pre-processing & Filtering Software (PaFS)
  - Main functionalities completed
    - Pre-processing
    - Cleaning
    - Filtering
    - Correlation
  - 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
  - To be achieved until M27:
    - Testing with additional datasets (derived from the DataPorts network)
    - Integration with on-demand component of Semantic Interoperability Layer
- Virtual Data Repository (VDR)
  - Initial 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
    - To be achieved until M27:
        - Perform additional tests to ensure that the system is working as expected
        - Enhance the architecture with auto-scaling mechanisms to improve the performance
        - Provide data sanitization algorithm to guarantee data integrity
- Virtual Data Container (VDC)
    - Main functionalities completed
        - An abstraction layer between the data consumers (e.g., data-intensive applications) and the VDR
        - Returns a portion of the data (Data Pond) via RESTful API
        - Able to transform the data to serve it in the right format (e.g., from JSON to Parquet)
    - 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
    - To be achieved until M27:
        - More complicated methods are expected to be implemented, highlighting the computational capabilities of VDCs
        - OpenAPI (Swagger) description of the API to be specified
        - Integration with Automatic Model Training Engine

Currently, the Data Abstraction and Virtualization component source code is stored in the following repository:

- https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav

The subcomponents are stored and managed in the following repositories:

- https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav/-/tree/main/PreProcessing%20&%20Filtering%20Software%20(PaFS)
- https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav/-/tree/main/Virtual%20Data%20Container%20(VDC)
- https://egitlab.iti.es/dataports/data_processing/data-abstraction-virtualization_dav/-/tree/main/Virtual%20Data%20Repository%20(VDR)

Finally, the code is managed inside the following subgroup:

- https://egitlab.iti.es/dataports/data_processing

5.4 EXAMPLE OF USE: DEMONSTRATION

5.4.1 Introduction

Data Abstraction & Virtualization component is considered as an internal framework of the DataPorts Platform, thus developers interact with VDC (DAV's third sub-component) via the aforementioned exposed RESTful APIs. Therefore, we shall look at a typical DAV process, from the starting point (where a dataset is

entered in PaFS) until the end (where a user – data recipient makes a RESTful request to VDC and gets a Parquet formatted data pond in response). A Greek weather dataset (Figure 28) is used for demonstration purposes. This dataset is free to use and can be accessed here: https://www.kaggle.com/spirospolitis/greek-weather-data.

```
Dataset length: 921561
NaN Values found:
Unnamed: 0                     0
STATION_REGION                 0
STATION_NAME                   0
YEAR                           0
MONTH                          0
DAY                            0
MEAN_TEMP                   1118
HIGH_TEMP                   1179
HIGH_TEMP_TIME             35581
LOW_TEMP                    1117
LOW_TEMP_TIME              35537
HEAT_DEG_DAYS               2098
COOL_DEG_DAYS               2129
RAIN                        3651
AVG_WIND_SPEED              3400
HIGHEST_WIND_SPEED          3334
HIGHEST_WIND_SPEED_TIME    40459
WIND_DIR                       0
ELEVATION                 170187
LAT                       170187
LONG                      170187
LAT_DEG                   170187
LONG_DEG                  170187
dtype: int64
```

Figure 28 – Greek weather dataset

As seen above, it is a "dirty" dataset, meaning that it contains multiple NaN values in different columns. Furthermore, NaN values are detected in both numerical and categorical columns. The dataset will now enter the Data Abstraction & Virtualization component.

```
rs0:PRIMARY> db.stats()
{
        "db" : "vdrDb",
        "collections" : 0,
        "views" : 0,
        "objects" : 0,
        "avgObjSize" : 0,
        "dataSize" : 0,
        "storageSize" : 0,
        "totalSize" : 0,
        "indexes" : 0,
        "indexSize" : 0,
        "scaleFactor" : 1,
        "fileSize" : 0,
        "fsUsedSize" : 0,
        "fsTotalSize" : 0,
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1622537773, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1622537773, 1)
}
```

Figure 29 – Initial status of VDR, the DAV's data lake

As already analysed, VDR consists of a modified MongoDB (named “vdrDb”), which (for now) contains nothing, as shown in Figure 29.

5.4.2 Store historical data in VDR that are preprocessed, cleaned and filtered

We now initiate the PaFS sub-component of DAV, for it to begin the process of pre-processing, cleaning, and filtering the incoming dataset. For that reason, an HTTP GET request is sent to PaFS, using Postman[41] tool. A successful example of such kind of requests is depicted in Figure 30. By the end of the process, NaN values are expected to be eliminated and replaced properly.

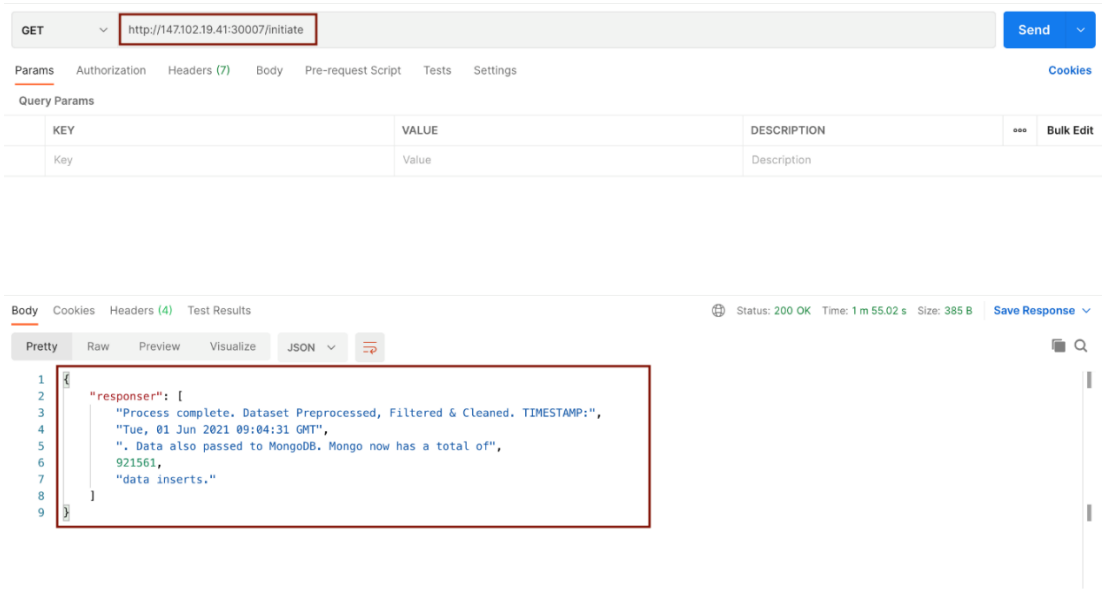


Figure 30 – Pre-processing, cleaning, and filtering functionality

After that, PaFS will proceed to store the dataset to VDR.

```
rs0:PRIMARY> db.stats()
{
        "db" : "vdrDb",
        "collections" : 1,
        "views" : 0,
        "objects" : 921561,
        "avgObjSize" : 528.7670821573396,
        "dataSize" : 487291121,
        "storageSize" : 91521024,
        "indexes" : 1,
        "indexSize" : 13447168,
        "totalSize" : 104968192,
        "scaleFactor" : 1,
        "fsUsedSize" : 57929732096,
        "fsTotalSize" : 124958425088,
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1622538703, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1622538703, 1)
}
```

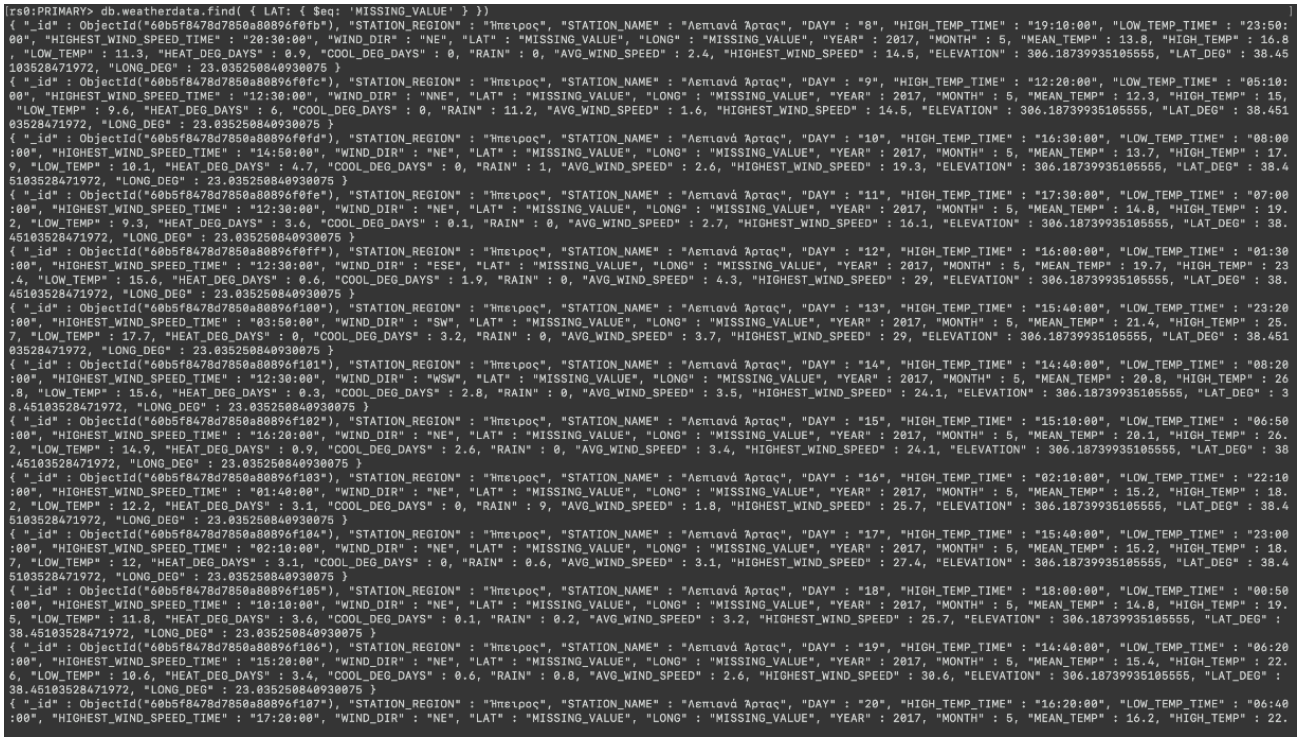
Figure 31 – Cleaned data is stored in VDR

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

We now check VDR again, to make sure that the cleaned dataset has been successfully passed in it from PaFS. The “vdrDb” database now has 921561 objects (Figure 31), which is the exact same number of the dataset total elements (as seen from the response in Figure 30). However, we need to make sure that the stored dataset has indeed been cleaned, containing zero NaN values. For example, searching for NaN values in the “LAT” column, which initially contained 170187 NaNs, returns nothing (Figure 32).

```
rs0:PRIMARY> db.weatherdata.find( { LAT: { $eq: NaN } })
rs0:PRIMARY>
```

Figure 32 – Verify that VDR contains zero NaN values



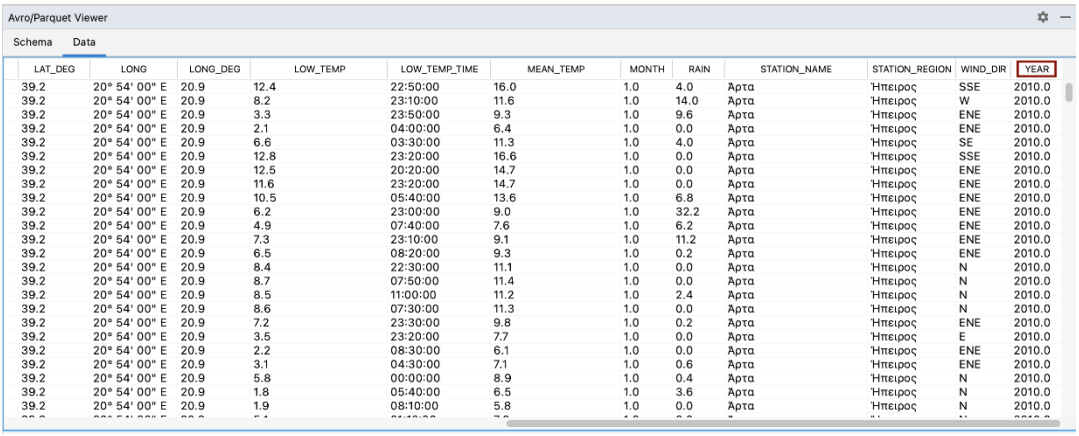
```
rs0:PRIMARY> db.weatherdata.find( { LAT: { $eq: 'MISSING_VALUE' } })
{ "_id" : ObjectId("60b5f8478d7850a80896f0fb"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "8", "HIGH_TEMP_TIME" : "19:10:00", "LOW_TEMP_TIME" : "23:50:
00", "HIGHEST_WIND_SPEED_TIME" : "20:30:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 13.8, "HIGH_TEMP" : 16.8
, "LOW_TEMP" : 11.3, "HEAT_DEG_DAYS" : 0.9, "COOL_DEG_DAYS" : 0, "RAIN" : 0, "AVG_WIND_SPEED" : 2.4, "HIGHEST_WIND_SPEED" : 14.5, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.45
103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f0fc"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "9", "HIGH_TEMP_TIME" : "12:20:00", "LOW_TEMP_TIME" : "05:10:
00", "HIGHEST_WIND_SPEED_TIME" : "12:30:00", "WIND_DIR" : "NNE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 12.3, "HIGH_TEMP" : 15,
 "LOW_TEMP" : 9.6, "HEAT_DEG_DAYS" : 6, "COOL_DEG_DAYS" : 0, "RAIN" : 11.2, "AVG_WIND_SPEED" : 1.6, "HIGHEST_WIND_SPEED" : 14.5, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.451
03528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f0fd"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "10", "HIGH_TEMP_TIME" : "16:30:00", "LOW_TEMP_TIME" : "08:00
:00", "HIGHEST_WIND_SPEED_TIME" : "14:50:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 13.7, "HIGH_TEMP" : 17.
9, "LOW_TEMP" : 10.1, "HEAT_DEG_DAYS" : 4.7, "COOL_DEG_DAYS" : 0, "RAIN" : 1, "AVG_WIND_SPEED" : 2.6, "HIGHEST_WIND_SPEED" : 19.3, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.4
5103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f0fe"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "11", "HIGH_TEMP_TIME" : "17:30:00", "LOW_TEMP_TIME" : "07:00
:00", "HIGHEST_WIND_SPEED_TIME" : "12:30:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 14.8, "HIGH_TEMP" : 19.
2, "LOW_TEMP" : 9.3, "HEAT_DEG_DAYS" : 3.6, "COOL_DEG_DAYS" : 0.1, "RAIN" : 0, "AVG_WIND_SPEED" : 2.7, "HIGHEST_WIND_SPEED" : 16.1, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.
45103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f0ff"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "12", "HIGH_TEMP_TIME" : "16:00:00", "LOW_TEMP_TIME" : "01:30
:00", "HIGHEST_WIND_SPEED_TIME" : "12:30:00", "WIND_DIR" : "ESE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 19.7, "HIGH_TEMP" : 23
.4, "LOW_TEMP" : 15.6, "HEAT_DEG_DAYS" : 0.6, "COOL_DEG_DAYS" : 1.9, "RAIN" : 0, "AVG_WIND_SPEED" : 4.3, "HIGHEST_WIND_SPEED" : 29, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.
45103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f100"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "13", "HIGH_TEMP_TIME" : "15:40:00", "LOW_TEMP_TIME" : "23:20
:00", "HIGHEST_WIND_SPEED_TIME" : "03:50:00", "WIND_DIR" : "SW", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 21.4, "HIGH_TEMP" : 25.
7, "LOW_TEMP" : 17.7, "HEAT_DEG_DAYS" : 0, "COOL_DEG_DAYS" : 3.2, "RAIN" : 0, "AVG_WIND_SPEED" : 3.7, "HIGHEST_WIND_SPEED" : 29, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.451
03528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f101"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "14", "HIGH_TEMP_TIME" : "14:40:00", "LOW_TEMP_TIME" : "08:20
:00", "HIGHEST_WIND_SPEED_TIME" : "12:30:00", "WIND_DIR" : "WSW", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 20.8, "HIGH_TEMP" : 26
.8, "LOW_TEMP" : 15.6, "HEAT_DEG_DAYS" : 0.3, "COOL_DEG_DAYS" : 2.8, "RAIN" : 0, "AVG_WIND_SPEED" : 3.5, "HIGHEST_WIND_SPEED" : 24.1, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 3
8.45103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f102"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "15", "HIGH_TEMP_TIME" : "15:10:00", "LOW_TEMP_TIME" : "06:50
:00", "HIGHEST_WIND_SPEED_TIME" : "16:20:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 20.1, "HIGH_TEMP" : 26.
2, "LOW_TEMP" : 14.9, "HEAT_DEG_DAYS" : 0.9, "COOL_DEG_DAYS" : 2.6, "RAIN" : 0, "AVG_WIND_SPEED" : 3.4, "HIGHEST_WIND_SPEED" : 24.1, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38
.45103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f103"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "16", "HIGH_TEMP_TIME" : "02:10:00", "LOW_TEMP_TIME" : "22:10
:00", "HIGHEST_WIND_SPEED_TIME" : "01:40:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 15.2, "HIGH_TEMP" : 18.
2, "LOW_TEMP" : 12.2, "HEAT_DEG_DAYS" : 3.1, "COOL_DEG_DAYS" : 0, "RAIN" : 9, "AVG_WIND_SPEED" : 1.8, "HIGHEST_WIND_SPEED" : 25.7, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.4
5103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f104"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "17", "HIGH_TEMP_TIME" : "15:40:00", "LOW_TEMP_TIME" : "23:00
:00", "HIGHEST_WIND_SPEED_TIME" : "02:10:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 15.2, "HIGH_TEMP" : 18.
7, "LOW_TEMP" : 12, "HEAT_DEG_DAYS" : 3.1, "COOL_DEG_DAYS" : 0, "RAIN" : 0.6, "AVG_WIND_SPEED" : 3.1, "HIGHEST_WIND_SPEED" : 27.4, "ELEVATION" : 306.18739935105555, "LAT_DEG" : 38.4
5103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f105"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "18", "HIGH_TEMP_TIME" : "18:00:00", "LOW_TEMP_TIME" : "00:50
:00", "HIGHEST_WIND_SPEED_TIME" : "10:10:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 14.8, "HIGH_TEMP" : 19.
5, "LOW_TEMP" : 11.8, "HEAT_DEG_DAYS" : 3.6, "COOL_DEG_DAYS" : 0.1, "RAIN" : 0.2, "AVG_WIND_SPEED" : 3.2, "HIGHEST_WIND_SPEED" : 25.7, "ELEVATION" : 306.18739935105555, "LAT_DEG" :
38.45103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f106"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "19", "HIGH_TEMP_TIME" : "14:40:00", "LOW_TEMP_TIME" : "06:20
:00", "HIGHEST_WIND_SPEED_TIME" : "15:20:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 15.4, "HIGH_TEMP" : 22.
6, "LOW_TEMP" : 10.6, "HEAT_DEG_DAYS" : 3.4, "COOL_DEG_DAYS" : 0.6, "RAIN" : 0.8, "AVG_WIND_SPEED" : 2.6, "HIGHEST_WIND_SPEED" : 30.6, "ELEVATION" : 306.18739935105555, "LAT_DEG" :
38.45103528471972, "LONG_DEG" : 23.035250840930075 }
{ "_id" : ObjectId("60b5f8478d7850a80896f107"), "STATION_REGION" : "Ηπειρος", "STATION_NAME" : "Λεπιανά Άρτας", "DAY" : "20", "HIGH_TEMP_TIME" : "16:20:00", "LOW_TEMP_TIME" : "06:40
:00", "HIGHEST_WIND_SPEED_TIME" : "17:20:00", "WIND_DIR" : "NE", "LAT" : "MISSING_VALUE", "LONG" : "MISSING_VALUE", "YEAR" : 2017, "MONTH" : 5, "MEAN_TEMP" : 16.2, "HIGH_TEMP" : 22.
```


Figure 33 – Verify that PaFS has replaced NaN values with “MISSING_VALUES”

However, searching the same “LAT” column for the value “MISSING_VALUE”, which PaFS replaced NaNs with, returns multiple results (Figure 33). This means that PaFS has successfully performed the expected functionalities, by cleaning & filtering the Greek weather dataset, before passing it to VDR for storage.

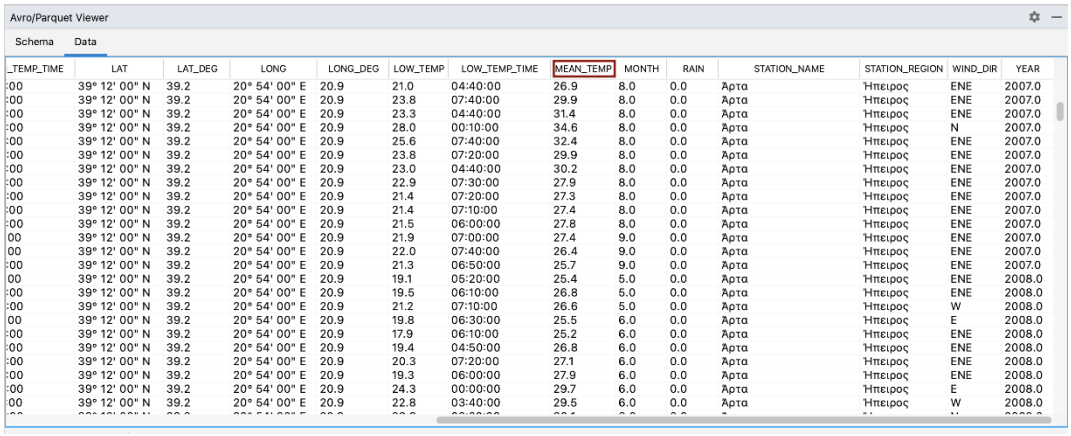
5.4.3 Request VDC to retrieve data ponds

Given that the dataset has been properly stored in VDR, we can now make RESTful requests through VDC, to acquire data ponds in a Parquet format. Using the “/yearly” path (Figure 34), any data consumer can retrieve all weather entries regarding a year (e.g., 2010). Similarly, the “/tempOver” method (Figure 35) returns all weather entries with mean temperature above a specific value (e.g., 25). Finally, via the “/rainNWind” path (Figure 36), VDC provides those weather rows that exceed particular rain and average wind speed threshold values (e.g., 0.8 and 2 respectively).



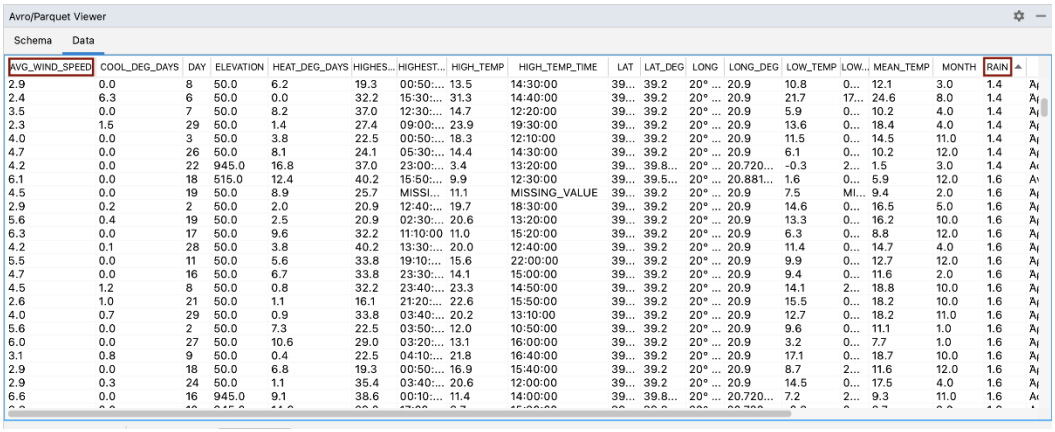
LAT_DEG	LONG	LONG_DEG	LOW_TEMP	LOW_TEMP_TIME	MEAN_TEMP	MONTH	RAIN	STATION_NAME	STATION_REGION	WIND_DIR	YEAR
39.2	20° 54' 00" E	20.9	12.4	22:50:00	16.0	1.0	4.0	Άρτα	Ήπειρος	SSE	2010.0
39.2	20° 54' 00" E	20.9	8.2	23:10:00	11.6	1.0	14.0	Άρτα	Ήπειρος	W	2010.0
39.2	20° 54' 00" E	20.9	3.3	23:50:00	9.3	1.0	9.6	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	2.1	04:00:00	6.4	1.0	0.0	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	6.6	03:30:00	11.3	1.0	4.0	Άρτα	Ήπειρος	SE	2010.0
39.2	20° 54' 00" E	20.9	12.8	23:20:00	16.6	1.0	0.0	Άρτα	Ήπειρος	SSE	2010.0
39.2	20° 54' 00" E	20.9	12.5	20:20:00	14.7	1.0	0.0	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	11.6	23:20:00	14.7	1.0	0.0	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	10.5	05:40:00	13.6	1.0	6.8	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	6.2	23:00:00	9.0	1.0	32.2	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	4.9	07:40:00	7.6	1.0	6.2	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	7.3	23:10:00	9.1	1.0	11.2	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	6.5	08:20:00	9.3	1.0	0.2	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	8.4	22:30:00	11.1	1.0	0.0	Άρτα	Ήπειρος	N	2010.0
39.2	20° 54' 00" E	20.9	8.7	07:50:00	11.4	1.0	0.0	Άρτα	Ήπειρος	N	2010.0
39.2	20° 54' 00" E	20.9	8.5	11:00:00	11.2	1.0	2.4	Άρτα	Ήπειρος	N	2010.0
39.2	20° 54' 00" E	20.9	8.6	07:30:00	11.3	1.0	0.0	Άρτα	Ήπειρος	N	2010.0
39.2	20° 54' 00" E	20.9	7.2	23:30:00	9.8	1.0	0.2	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	3.5	23:20:00	7.7	1.0	0.0	Άρτα	Ήπειρος	E	2010.0
39.2	20° 54' 00" E	20.9	2.2	08:30:00	6.1	1.0	0.0	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	3.1	04:30:00	7.1	1.0	0.6	Άρτα	Ήπειρος	ENE	2010.0
39.2	20° 54' 00" E	20.9	5.8	00:00:00	8.9	1.0	0.4	Άρτα	Ήπειρος	N	2010.0
39.2	20° 54' 00" E	20.9	1.8	05:40:00	6.5	1.0	3.6	Άρτα	Ήπειρος	N	2010.0
39.2	20° 54' 00" E	20.9	1.9	08:10:00	5.8	1.0	0.0	Άρτα	Ήπειρος	N	2010.0

Figure 34 – Output of VDC method #1



_TEMP_TIME	LAT	LAT_DEG	LONG	LONG_DEG	LOW_TEMP	LOW_TEMP_TIME	MEAN_TEMP	MONTH	RAIN	STATION_NAME	STATION_REGION	WIND_DIR	YEAR
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.0	04:40:00	26.9	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	23.8	07:40:00	29.9	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	23.3	04:40:00	31.4	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	28.0	00:10:00	34.6	8.0	0.0	Άρτα	Ήπειρος	N	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	25.6	07:40:00	32.4	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	23.8	07:20:00	29.9	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	23.0	04:40:00	30.2	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	22.9	07:30:00	27.9	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.4	07:20:00	27.3	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.4	07:10:00	27.4	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.5	06:00:00	27.8	8.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.9	07:00:00	27.4	9.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	22.0	07:40:00	26.4	9.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.3	06:50:00	25.7	9.0	0.0	Άρτα	Ήπειρος	ENE	2007.0
00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	19.1	05:20:00	25.4	5.0	0.0	Άρτα	Ήπειρος	ENE	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	19.5	06:10:00	26.8	5.0	0.0	Άρτα	Ήπειρος	ENE	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	21.2	07:10:00	26.6	5.0	0.0	Άρτα	Ήπειρος	W	2008.0
00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	19.8	06:30:00	25.5	6.0	0.0	Άρτα	Ήπειρος	E	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	17.9	06:10:00	25.2	6.0	0.0	Άρτα	Ήπειρος	ENE	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	19.4	04:50:00	26.8	6.0	0.0	Άρτα	Ήπειρος	ENE	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	20.3	07:20:00	27.1	6.0	0.0	Άρτα	Ήπειρος	ENE	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	19.3	06:00:00	27.9	6.0	0.0	Άρτα	Ήπειρος	ENE	2008.0
:00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	24.3	00:00:00	29.7	6.0	0.0	Άρτα	Ήπειρος	E	2008.0
00	39° 12' 00" N	39.2	20° 54' 00" E	20.9	22.8	03:40:00	29.6	6.0	0.0	Άρτα	Ήπειρος	W	2008.0

Figure 35 – Output of VDC method #2



AVG_WIND_SPEED	COOL_DEG_DAYS	DAY	ELEVATION	HEAT_DEG_DAYS	HIGHES...	HIGHEST...	HIGH_TEMP	HIGH_TEMP_TIME	LAT	LAT_DEG	LONG	LONG_DEG	LOW_TEMP	LOW...	MEAN_TEMP	MONTH	RAIN
2.9	0.0	8	50.0	6.2	19.3	00:50:...	13.5	14:30:00	39...	39.2	20° ...	20.9	10.8	0...	12.1	3.0	1.4
2.4	6.3	6	50.0	0.0	32.2	15:30:...	31.3	14:40:00	39...	39.2	20° ...	20.9	21.7	17...	24.6	8.0	1.4
3.5	0.0	7	50.0	8.2	37.0	12:30:...	14.7	12:20:00	39...	39.2	20° ...	20.9	5.9	0...	10.2	4.0	1.4
2.3	1.5	29	50.0	1.4	27.4	09:00:...	23.9	19:30:00	39...	39.2	20° ...	20.9	13.6	0...	18.4	4.0	1.4
4.0	0.0	3	50.0	3.8	22.5	00:50:...	18.3	12:10:00	39...	39.2	20° ...	20.9	11.5	0...	14.5	11.0	1.4
4.7	0.0	26	50.0	8.1	24.1	05:30:...	14.4	14:30:00	39...	39.2	20° ...	20.9	6.1	0...	10.2	12.0	1.4
4.2	0.0	22	945.0	16.8	37.0	23:00:...	3.4	13:20:00	39...	39.8...	20° ...	20.720...	-0.3	2...	1.5	3.0	1.4
6.1	0.0	18	515.0	12.4	40.2	15:50:...	9.9	12:30:00	39...	39.5...	20° ...	20.881...	1.6	0...	5.9	12.0	1.6
4.5	0.0	19	50.0	8.9	25.7	MISSI...	11.1	MISSING_VALUE	39...	39.2	20° ...	20.9	7.5	MI...	9.4	2.0	1.6
2.9	0.2	2	50.0	2.0	20.9	12:40:...	19.7	18:30:00	39...	39.2	20° ...	20.9	14.6	0...	16.5	5.0	1.6
5.6	0.4	19	50.0	2.5	20.9	02:30:...	20.6	13:20:00	39...	39.2	20° ...	20.9	13.3	0...	16.2	10.0	1.6
6.3	0.0	17	50.0	9.6	32.2	11:10:00	11.0	15:20:00	39...	39.2	20° ...	20.9	6.3	0...	8.8	12.0	1.6
4.2	0.1	28	50.0	3.8	40.2	13:30:...	20.0	12:40:00	39...	39.2	20° ...	20.9	11.4	0...	14.7	4.0	1.6
5.5	0.0	11	50.0	5.6	33.8	19:10:...	15.6	22:00:00	39...	39.2	20° ...	20.9	9.9	0...	12.7	12.0	1.6
4.7	0.0	16	50.0	6.7	33.8	23:30:...	14.1	15:00:00	39...	39.2	20° ...	20.9	9.4	0...	11.6	2.0	1.6
4.5	1.2	8	50.0	0.8	32.2	23:40:...	23.3	14:50:00	39...	39.2	20° ...	20.9	14.1	2...	18.8	10.0	1.6
2.6	1.0	21	50.0	1.1	16.1	21:20:...	22.6	15:50:00	39...	39.2	20° ...	20.9	15.5	0...	18.2	10.0	1.6
4.0	0.7	29	50.0	0.9	33.8	03:40:...	20.2	13:10:00	39...	39.2	20° ...	20.9	12.7	0...	18.2	11.0	1.6
5.6	0.0	2	50.0	7.3	22.5	03:50:...	12.0	10:50:00	39...	39.2	20° ...	20.9	9.6	0...	11.1	1.0	1.6
6.0	0.0	27	50.0	10.6	29.0	03:20:...	13.1	16:00:00	39...	39.2	20° ...	20.9	3.2	0...	7.7	1.0	1.6
3.1	0.8	9	50.0	0.4	22.5	04:10:...	21.8	16:40:00	39...	39.2	20° ...	20.9	17.1	0...	18.7	10.0	1.6
2.9	0.0	18	50.0	6.8	19.3	00:50:...	16.9	15:40:00	39...	39.2	20° ...	20.9	8.7	2...	11.6	12.0	1.6
2.9	0.3	24	50.0	1.1	35.4	03:40:...	20.6	12:00:00	39...	39.2	20° ...	20.9	14.5	0...	17.5	4.0	1.6
6.6	0.0	16	945.0	9.1	38.6	00:10:...	11.4	14:00:00	39...	39.8...	20° ...	20.720...	7.2	2...	9.3	11.0	1.6

Figure 36 – Output of VDC method #3

6 CONCLUSIONS

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.

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. An initial version of this component, which provides the core functionality, is currently 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 applications.
- 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 integration with Blockchain and IDS and support Linked Data. Also, additional extensions might be included if new requirements from the pilots are identified.

Regarding the Data Model, the methodology and guidelines have been defined and the first version, which will be integrated in the Semantic Interoperability Components and other components of the DataPorts Platform, is expected by M24.

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 can be retrieved in a selected format, such as Parquet. The three sub-components responsible for the complete process are 1) the (Pre)Processing and Filtering Software, where the initial pre-processing and cleaning 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 selected data ponds to potential data recipients is achieved.

This component will be updated and extended in the following months to provide integration with the Semantic Interoperability layer and the Automatic Models Training Engine. Moreover, additional features will be implemented towards the completion of the project’s requirements.

7 REFERENCES AND ACRONYMS

7.1 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, “D2.4 Platform architecture and specifications” December 2021.

[4] DataPorts Consortium, “D2.1 Industrial

[5] Data Platforms and seaport community requirements and challenges” March 2021.

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

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

[8] 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.

7.2 ACRONYMS

Acronym List	
API	Application Programming Interface
CEF	Connecting Europe Facility
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
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
UI	User Interface
UML	Unified Modelling Language
URN	Uniform Resource Name
VDC	Virtual Data Container
VDR	Virtual Data Repository
WP	Work Package

Table 1 – Acronyms

8 ANNEX A: METADATA REGISTRY

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

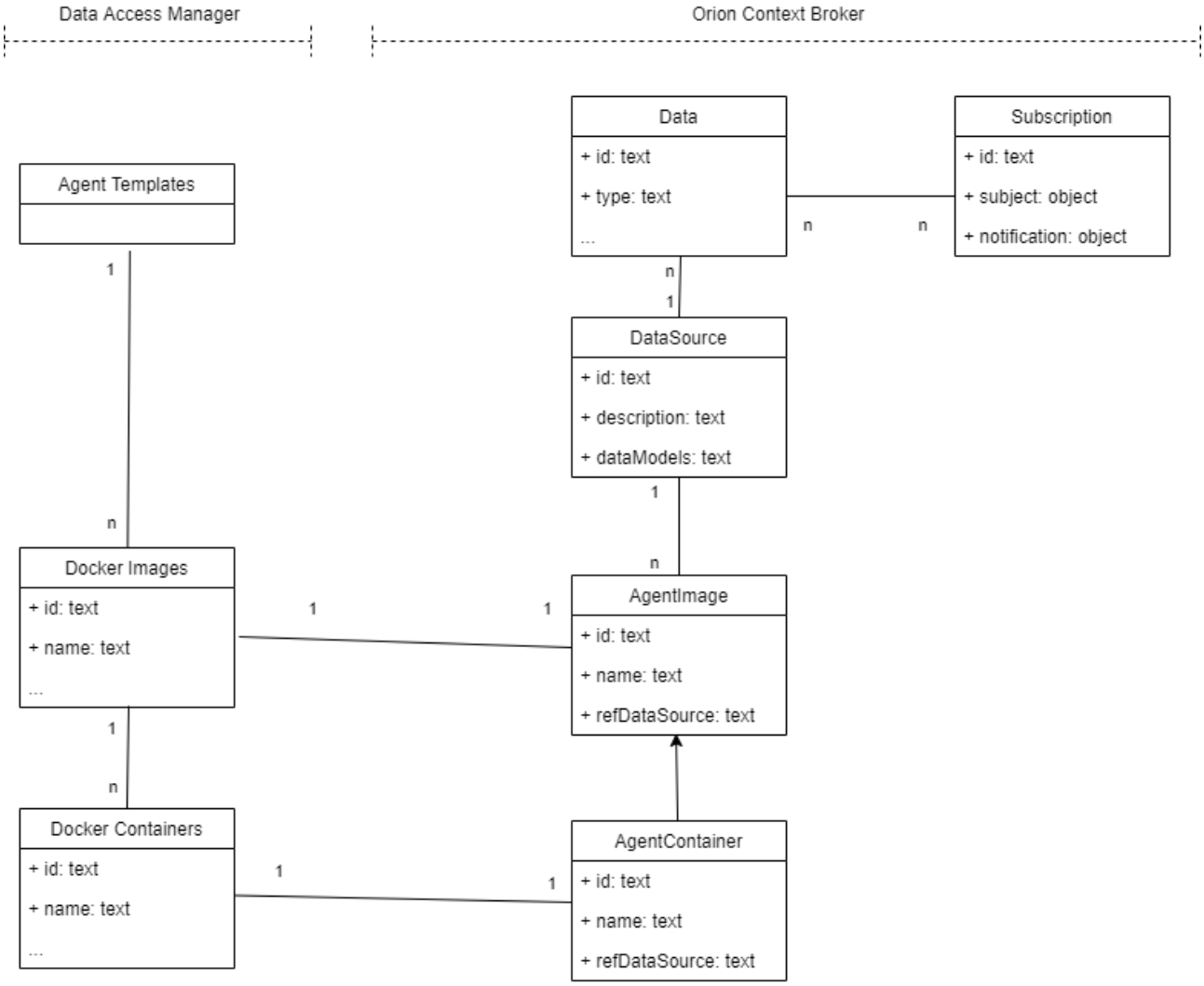


Figure 37 – Metadata registry data model

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 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 2 – 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:

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 3 – 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:

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 4 – Publish/subscribe agent metadata