Deliverable
D3.1 Data Manager Layer. Initial

Project Acronym:	URBANAGE	
Project title:	Enhanced URBAN planning for AGE-friendly cities through disruptive technologies	
Grant Agreement No.	101004590	
Website:	www.urbanage.eu	
Version:	1.0	
Date:	31/01/2022	
Responsible Partner:	ENG	
Contributing Partners:	ENG, TEC, ATC, CIC	
Reviewers:	Athanasios Dalianis (ATC) Christoph Fink (UH)	
Dissemination Level:	Public	x
	Confidential – only consortium members and European Commission	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004590

Revision History

Revision	Date	Author	Organization	Description
0.1	12/08/2021	Giuseppe Ciulla	ENG	Table of Content
0.2	17/09/2021	Giuseppe Ciulla	ENG	Initial draft
0.3	25/10/2021	Giuseppe Ciulla, Marco Martorana	ENG	Second draft
0.4	15/12/2021	Giuseppe Ciulla, Marco Martorana	ENG	Content for section 3 and anexes
0.5	12/01/2022	Giuseppe Ciulla, Marco Martorana	ENG	Content consolidation
0.6	14/01/2022	Giuseppe Ciulla	ENG	Document ready for internal review
0.7	21/01/2022	Athanasios Dalianis Christoph Fink	ATC UH	Internal review
1.0	28/01/2022	Giuseppe Ciulla Claudia Vicari	ENG	Final version

Table of Contents

Table of Figures

Table of Tables

List of abbreviations

Abbreviation	Explanation
API	Application Programming Interface
BDS	Big Data Analytic System
CI/CD	Continuous Integration and Continuous Delivery
CIDB	Context Information Data Bridge
CityGML	City Geography Markup Language
CKAN	Comprehensive Knowledge Archive Network
CSV	Comma-Separated Values
DAG	Directed Acyclic Graphs
DBMS	Database Management Systems
DCAT	Data Catalog Vocabulary
DACT-AP	DCAT Application Profile
DL	Deep Learning
DML	Data Management Layer
ETSI	European Telecommunications Standards Institute
GE	Generic Enabler
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IT	Information Technology
JSON	JavaScript Object Notation
JSON-LD	JSON Linked Data
KML	Keyhole Markup Language
LAM	Liikenteen Automaattinen Mittaus (Automatic Traffic Measurement)
LOD	Linked Open Data

ML	Machine Learning
N3	Notation-3
NGSI	Next Generation Service Interfaces
ODMS	Open Data Management Systems
OMA	Open Mobile Alliance
RDF	Resource Description Format
RDFS	RDF Schema
REST	Representational State Transfer
SAREF	Smart Applications REFerence
SOSA	Sensor, Observation, Sample, and Actuator
SSN	Semantic Sensor Network
Turtle	Terse RDF Triple Language
UI	User Interface
URI	Uniform Resource Identifier
WMS	Web Map Service

1 Executive Summary

This document describes the first version of the design of the Data Management Layer (DML) of the URBANAGE Platform. The purpose of the DML is to enable the URBANAGE Platform to collect data from heterogeneous sources, aggregate different types of information and harmonise them using common ontologies.

For this purpose, this document presents and describes data formats (such as JSON-LD and NSGI-LD), ontologies (such as SAREF and some of its extensions) and data models (as the ones offered by the Smart Data Models initiatives) to set the ground for a uniform and interoperable representation of the collected data.

Then, the initial design of the Data Management Layer is presented highlighting the connection with the possible typologies of data sources, as well as the other foreseen components of the URBANAGE Platform. The logical components of the DML and the relations among them are so described and the baseline technical tool for their realisation are introduced; among them

- the Orion Context Broker for the management of context information (that is also one of the building blocks promoted by the Connecting Europe Facility Programme[1] and represents an interoperability point with future initiatives) and
- Idra for the harmonisation of metadata of datasets managed by Open Data Portals (or Open Data Management Systems) and for the realisation of catalogues to search and discover datasets managed by the DML.

This document also presents an overview of the processes adopted by the DML to collect and harmonise both static and real-time data, as well as an overview of the main APIs exposed by the DML to access the managed data.

Finally, annexes report a summary of the dataset identified by the use cases, the overview of the main components of the URBANAGE platform that interact with the Data Management Layer, and the technical details of the baseline tools for its implementation.

[1] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/CEF+Digital+Home

2 Introduction

The Data Management Layer (DML) is part of the URBANAGE Platform, whose main objective is to offer an ecosystem that integrates capabilities for multidimensional analysis of Big Data, modelling and simulation with Artificial Intelligence algorithms and visualization through Urban Digital twins.

Considering the conceptual drawing of URBANAGE's software architecture (Figure 1), DML represents the interface of the URBANAGE Platform with the possible data sources that will offer the data to feed the capabilities of the platform itself. In a few words, within the overall framework of URBANAGE's software architecture, the DML aims to provide the mechanisms and procedures that allow the URBANAGE Platform to collect, aggregate and harmonise heterogeneous data coming from diverse data sources.

In particular, the DML will allow the URBANAGE Platform to collect data from both open and private data sources, by supporting different technologies and systems (e.g. IoT devices and platforms, open data portals, databases and city legacy systems). Leveraging common ontologies (e.g. SAREF) and standard data format (e.g. NGSI-LD), the DML will allow to harmonise and aggregate the collected data, following a uniform representation, through the adoption and definition of common data models. The DML will make the managed data available to other components of the URBANAGE through a set of APIs.

From a technical viewpoint, the DML reuses existing open-source tools components, and its design integrates them to fit the specific needs of the URBANAGE Platform related to data collection aggregation and harmonisation. Furthermore, to foster interoperability with both data sources and with the other components of the whole URBANAGE Platform, the design of the DML followed the no-vendor lock-in principle, by making use of open API and common data models for interoperability.

Deliverable “D5.1 System Architecture & Implementation Plan” provides details about the overall architecture of the URBANAGE Platform.

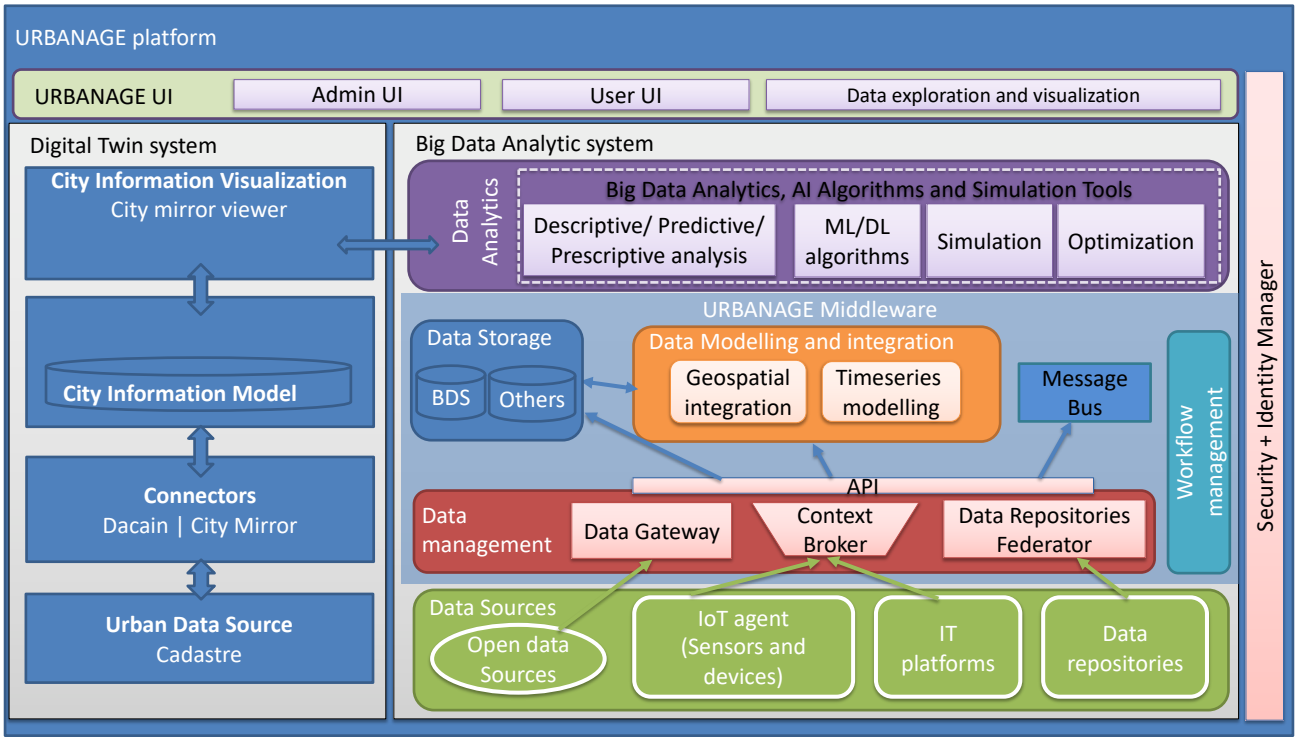
This document is organized in six sections.

- Section 1 is the executive summary of this document
- Section 2 is the introduction (this section)
- Section 3 introduces the candidate data formats, ontologies and data models for the implementation of the Data Management Layer.
- Section 4 describes the Data Management Layer, its design, the components and the processes for data collection, harmonisation and aggregation.
- Section 5 presents the conclusion, while
- Section 6 contains the references cited in this deliverable.

Furthermore, three annexes are included in this document.

- "Annex 1 – Overview of Initial datasets from pilot sites" offers an overview of the initial datasets identified by within the three pilot sites of the project (Santander, Flanders, and Helsinki).
- "Annex 2 – Main components of URBANAGE Platform interacting with the Data Management Layer" briefly reports the components of the URBANAGE Platform that have a direct interaction with the Data Management Layer
- "Annex 3 – Data Management Layer baseline tools" describe the baseline technological tools for the implementation of the Data Management Layer

Figure 1: Overview URBANAGE Platform[2]



[2] Image from deliverable "D5.1 System Architecture & Implementation Plan"

3 Data Interoperability

The Data Management Layer will represent an interoperability point between the whole URBANAGE platform and data sources. Data sources include data repositories, sensors, (legacy) IT Systems and Open Data Management Systems that will be interconnected to the platform itself to get the data they offer and to achieve semantic interoperability. Semantic interoperability is a fundamental building block, together with the adoption of open standards to enable collaboration between organizations and systems and to avoid information silos.

In a few words, to achieve data interoperability, it is necessary to establish a common data format to concretely represent the information. The data format defines how the information is structured and how machines can read it. But the data format doesn't establish any relation among the different pieces of the information to represent. Nevertheless, this is what ontologies do. An ontology defines the different concepts belonging to one or more domains (e.g. building, smart cities, environment, etc.), together with their relative properties and relations.

Finally, data models offer a well-defined way to represent the information; a data model leverages a data format (to structure the information and the relations among the different pieces of the information) and one or more ontologies to properly represent the parts of the information it contains and their relations, among the contained parts, as well as with other data models.

This section briefly describes the candidate data formats, ontologies and data models for the implementation of the Data Management Layer.

3.1 Formats

3.1.1 JSON / JSON-LD

JSON (JavaScript Object Notation) is a lightweight data interchange format. It is promoted as a low-overhead alternative to XML. It is a text-based format that is completely language independent based on a subset of the JavaScript Programming Language Standard [1]. JSON defines a small set of structuring rules for the portable representation of structured data. It is agnostic about numbers offering only the representation of numbers that humans use: a sequence of digits.
JSON is built on two data structures:

- A collection of name/value pairs.
- An ordered list of values.

These are universal data structures that in various languages are represented respectively as objects, records, dictionaries, hash tables (i.e. collections in JSON) and array, vectors, list, or sequences (i.e. ordered list in JSON).

A typical JSON object is an unordered set of name/value pairs that begins with left brace "{" and ends with right brace "}". Each name is followed by ":" colon and the name/value pairs are separated by "," comma. Figure 2 depicts an example of a JSON object.

Figure 2: Example of JSON object representation

```
{
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

JSON-LD (JavaScript Object Notation for Linked Data) is a lightweight Linked Data format, easy for humans to read and write, based on the JSON format [2] [3]. JSON-LD is designed around the concept of a "context" to provide additional mappings from JSON to an RDF model (see section 3.1.3 about RDF). The context links object properties in a JSON document to concepts in an ontology. In order to map the JSON-LD syntax to RDF, JSON-LD allows values to be coerced to a specified type or to be tagged with a specific spoken language. A context can be embedded directly in a JSON-LD document or put into a separate file and referenced from different documents via a specific HTTP link. Figure 3 depicts an example of a JSON-LD object representation.

Figure 3: Example of JSON-LD object presentation

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

3.1.2 NGSI / NGSI-LD

The FIWARE Foundation[3] has adopted the NGSI specifications [4] initially issued by OMA (Open Mobile Alliance)[4] as the reference standard for implementing the information model used in its architecture. The FIWARE NGSI information model was implemented in the FIWARE ecosystem and later evolved into NGSI-LD to support linked data. Indeed, context definition, with all terms used, is the main new element in NGSI-LD as it is based on JSON-LD. Relevant benefits of these independent open standards are mainly devoted to achieve interoperability avoiding "vendor lock-in".

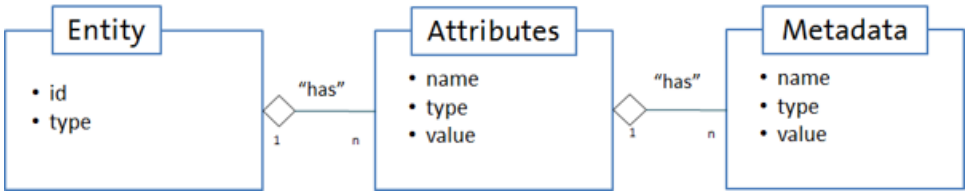
[3] A non-profit organization defining and promoting the adoption of open standards for the development of smart solutions across different domains such as Smart Cities, Smart Energy, Smart AgriFood and Smart Industry. - https://www.fiware.org/

[4] A non-profit non-governmental organization developing open, international technical standards for the mobile phone industry. Recently renamed "OMA SpecWorks" - https://omaspecworks.org/

NGSI Information model

The FIWARE NGSI information model is composed of three main elements that are respectively *Entities, Attributes and Metadata,* depicted in Figure 4.

Figure 4: NGSI information model



Entities are the "central" elements in the FIWARE NGSI information model. An entity in NGSI is a generic concept and can be anything; it depends completely on the data model of the user. Indeed, an entity represents a generic thing, any physical or logical object (e.g., a sensor, a person, a room, an issue in a ticketing system, etc.). Each entity has the mandatory field "id" with which it can be recognized.
Furthermore, the "type" field enables entities to have an entity type. Entity types are semantic types that are used to categorise the type of thing represented by the entity.
At the end, each entity is uniquely identified by the combination of its id and type; both fields are mandatory to define an NGSI entity.

Attributes are properties of context entities. They describe the entity they belong to. In the NGSI data model, attributes have a "name", a "type", a "value" and can be associated to metadata.
The attribute "name" describes what kind of property the attribute value represents, for example, "temperature". The attribute "type" represents the NGSI value type of the attribute value, which is usually similar or equivalent to the JSON datatype. The attribute "value", finally, contains the actual data.

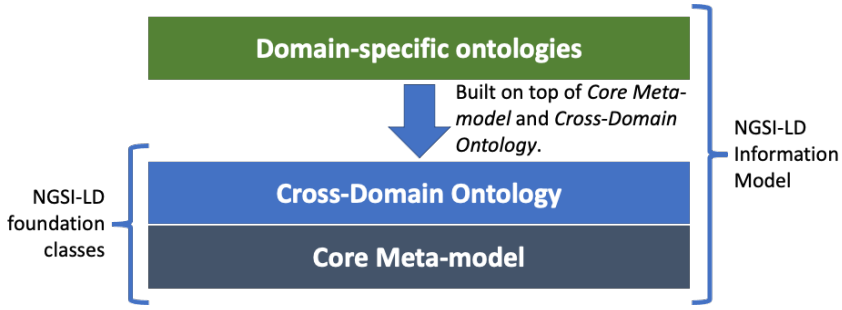
Metadata is used as an optional part of an attribute. Like attributes, metadata has "name", "type" and "value".

NGSI-LD Information model

The NGSI-LD Information Model is an evolution of NGSI that support linked data [5]. It prescribes the structure of context information that shall be supported by an NGSI-LD system specifying the data representation mechanisms to be used by the NGSI-LD API. Furthermore, the structure of the Context Information Management vocabularies to be used are defined. The NGSI-LD Context Information Management APIs allow users to provide, consume and subscribe to context information in multiple scenarios and involving multiple stakeholders. These APIs enables close to real-time access to information coming from many different sources typology not only IoT data sources but also IT systems in general.

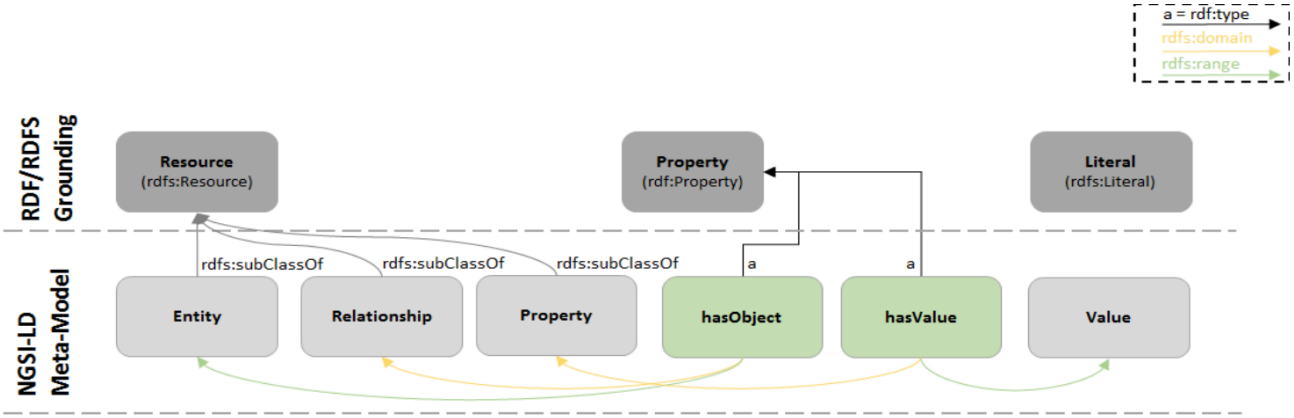
The NGSI-LD Information Model, which is based on JSON-LD, is defined at two levels: the foundation classes corresponding to the *Core Meta-model*, and the *Cross-Domain Ontology*. The *domain-specific ontologies* or vocabularies can be developed on top of these two nested levels.

Figure 5: NGSI-LD information model structure



A graphical representation of the NGSI-LD core Meta-Model in terms of classes and their relationships has been reported in Figure 6.

Figure 6: NGSI-LD Core Meta Model



Implementations have to support the NGSI-LD Meta-model as follows:

- An NGSI-LD Entity is a subclass of *rdfs:Resource*
- An NGSI-LD Relationship is a subclass of *rdfs:Resource*
- An NGSI-LD Property is a subclass of *rdfs:Resource*
- An NGSI-LD Value have to be either a *rdfs:Literal* or a node object to represent complex data structures
- An NGSI-LD Property shall have a value, stated through *hasValue*, which is of type *rdf:Property*
- An NGSI-LD Relationship shall have an object stated through *hasObject* which is of type *rdf:Property*

NGSI-LD API for Context Information Management

Concerning **interoperability** with external systems and applications, NGSI-LD provides a simple and consistent way to manage context information (Entity types, entities, attributes) through standard HTTP operations (GET, POST, PUT, PATCH, DELETE). Indeed, NGSI-LD provides APIs that allow to execute different operations, such as:

- Subscription / Notification
- Mechanisms devoted to federation
- Data Gathering

- Geo-queries
- Temporal operations
- Data “renderings” in multiple formats (key value, normalized, GeoJSON)

Table 1 reports an overview of the most relevant APIs defined by NGSI-LD. All resource URIs have the following root path: *{apiRoot}/{apiName}/{apiVersion}/*

Table 1: Resources and HTTP methods defined for main NGSI-LD APIs

Resource Name	Resource URI	HTTP Method	Description
Entity List	/entities/	POST	This API is in charge of creating a new entity
		GET	This API is in charge of performing queries against the existing entities
Entity by Id	/entities/{entityId}	GET	This API is in charge of retrieving an entity by its Id
		DELETE	This API is in charge of deleting and entity by its Id
Entity Attribute List	/entities/{entityId}/attrs/	POST	This API is in charge of appending a new attribute to an existing entity
		PATCH	This API is in charge of updating the attributes of an existing entity
Attribute by Id	/entities/{entityId}/attrs/{attrId}	PATCH	This API is in charge of updating a specific attribute (by its Id) of an existing entity
		DELETE	This API is in charge of deleting a specific attribute (by its Id) of an existing entity
Subscriptions List	/subscriptions/	POST	This API is in charge of creating a subscription
		GET	This API is in charge of retrieving the existing subscriptions
Subscription by Id	/subscriptions/{subscriptionId}	GET	This API is in charge of retrieving a specific subscription by its Id
		PATCH	This API is in charge of updating a specific subscription by its Id
		DELETE	This API is in charge of deleting a specific subscription by its Id
Entity Types	/types/	GET	This API is in charge of retrieving the available entity types
Entity Type	/types/{type}	GET	This API is in charge of retrieving details about a specific available entity type

Attributes	/attributes/	GET	This API is in charge of retrieving the available attributes
Attribute	/attributes/{attrId}	GET	This API is in charge of retrieving the details about a specific available attribute by its Id

3.1.3 RDF

RDF (Resource Description Framework) [6] is a standard model for data interchange on the Web. RDF was created by the W3C to represent and use structured metadata and to ensure interoperability between different resources that share information on the Web. It allows the simple representation of data semantics. It is used also as a standard for Open Data published on the World Wide Web. Indeed, the information encoded in RDF can be easily manipulated by agents and automatic machines. Concepts represented with RDF can always be accessed on the Web because they are identified by a unique URIs Indeed, RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

RDFS (RDF Scheme) [7] is an extension of RDF that deals with defining simple schemes to represent data and exposes the syntax to define the vocabularies for metadata. Resources can be represented by class instances and properties and constraints by subclasses, types and collections. In this way, the RDF triples can be connected through simple constructs, such as inheritance and extension relationships between classes, while common types of constraints can be those of domain and range. RDF enables effective data integration from multiple sources, detaching data from its schema.

It provides basic elements for the description of ontologies. Indeed, RDF is one of the main languages of the semantic web allowing to encode the relationships between subjects and objects through predicates. The set of relationships makes it possible to obtain a representation of knowledge. In RDF to define a relationship between things a "triple" is generally used to formalize this relation.

A brief description about each element of this triple is described below:

- **Subject**. A subject is always a uniquely identified resource (URI). It can be a person, a thing, an abstract concept, etc.
- **Predicate**. A predicate allows to create a relationship between a subject and an object. The same subject can have different relations with other objects.
- **Object**. An object can be either a uniquely identifiable resource (URI) or a piece of data.

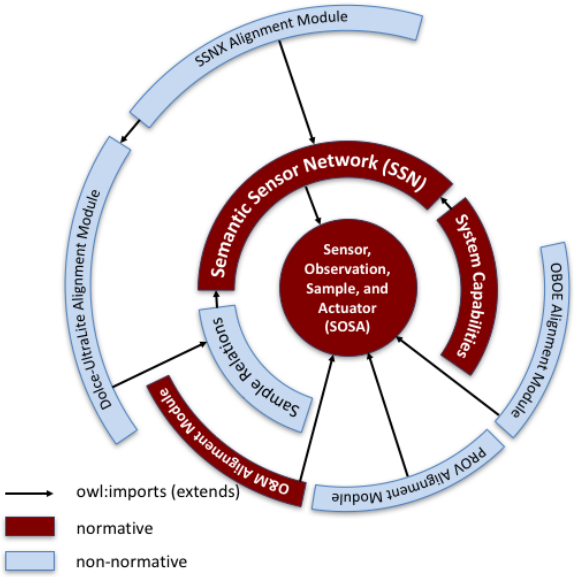
3.2 Ontologies

3.2.1 Semantic Sensor Network ontology

The Semantic Sensor Network (SSN) [8] is an ontology for describing sensors and their observations, the involved procedures, and the observed properties, etc. SSN follows a horizontal and vertical modularization architecture by including a lightweight but self-contained core ontology named SOSA (Sensor, Observation, Sample, and Actuator) for its elementary classes and properties [9].

Ontology modularization is a common method used in ontology engineering to segment an ontology into smaller parts. In general, ontology modularization aims at providing users of ontologies with the knowledge they require, reducing the scope as much as possible to what is strictly necessary in a given use case. SSN strongly uses two main approaches for ontology modularization: vertical and horizontal. With the vertical modularisation, modules are stacked on each other by using *owl:import* statement (with this approach it is mandatory to respect the chain of "import", otherwise inconsistencies may occur). With the horizontal modularisation, modules do not depend on each other; concept of different modules can be connected making use of a directional property (e.g. *subClassOf*).

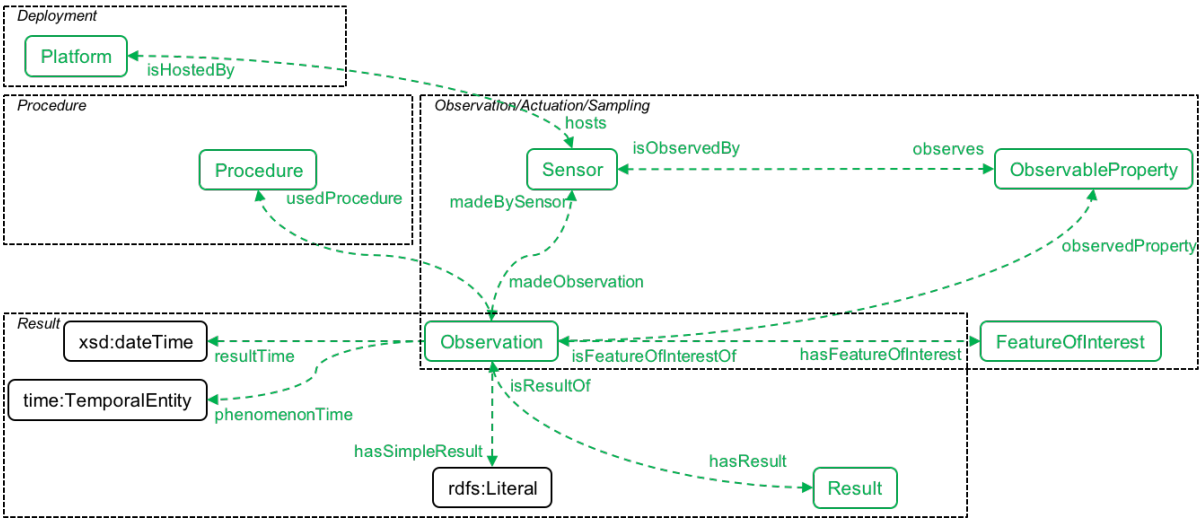
Figure 7: SOSA and SSN ontologies and their modules[5]



In SOSA/SSN several conceptual modules have been defined to cover key sensor, actuation and sampling concepts. Figure 8 provides an overview of the main classes and properties inside the ontology modules, from the perspectives of Observation, Actuation and Sampling.

[5] Image from SSN web page https://www.w3.org/TR/vocab-ssn/

Figure 8: Overview of SOSA classes and properties (observation perspective)



3.2.2 Smart Applications REFerence ontology

SAREF (Smart Applications REFerence) ontology [10] promoted by ETSI (European Telecommunications Standards Institute)[6] aims to define a common background with standardized interfaces and data models to ensure interoperability among platforms and systems.

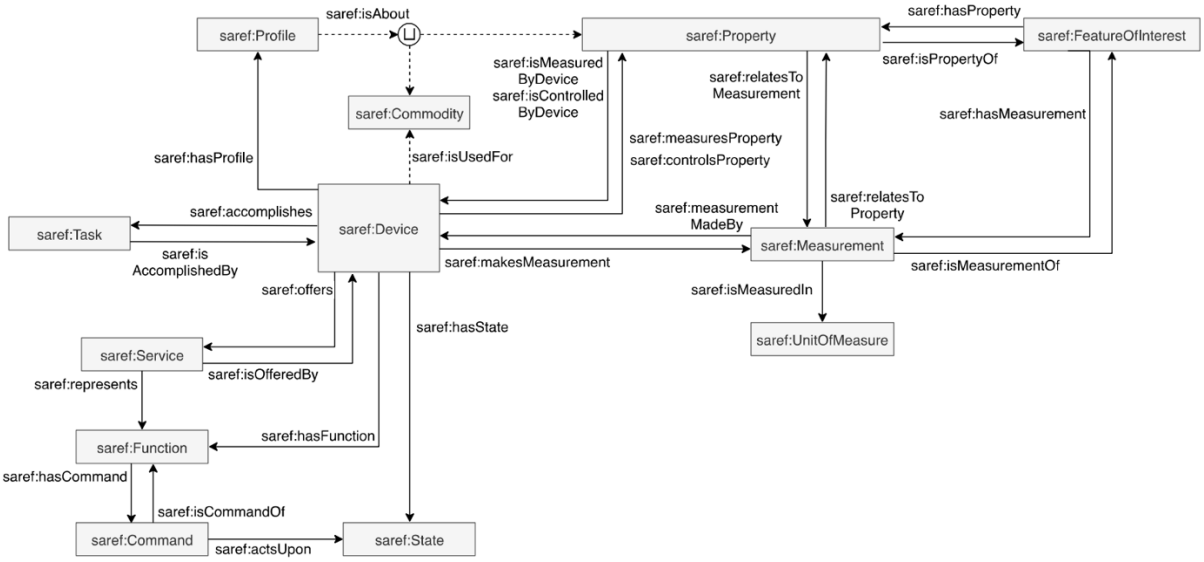
SAREF has been designed following principles such as modularity, extensibility, maintainability and reuse and alignment.

- ***Modularity*** to allow separation and recombination of different parts of the ontology based on specific needs.
- ***Extensibility*** to allow further growth of the ontology.
- ***Maintainability*** to facilitate the process of identifying and correcting defects, develop new requirements.
- ***Reuse and alignment*** of concepts and relationships that are defined in existing assets.

SAREF focuses on the concept of *device*, which is defined as a tangible object designed to accomplish a particular task in a specific domain/site. In order to accomplish this task, a device performs one or more functions. Typical examples of devices are elevator failure sensors, escalator operation sensors, temperature/humidity sensors, light switches, etc. Figure 9 depicts the *saref:Device* class and its properties.

[6] A not-for-profit organization in the field of information and communications - https://www.etsi.org/

Figure 9: SAREF main classes and their relationships[7]



SAREF includes different extensions, each of them addressing a specific domain. Currently, the available extensions are:

- SAREF4ENER: Energy domain
- SAREF4ENVI: Environment domain
- SAREF4BLDG: Building domain
- SAREF4CITY: Smart Cities domain
- SAREF4INMA: Industry and Manufacturing domains
- SAREF4AGRI: Smart Agriculture and Food Chain domains
- SAREF4AUTO: Automotive domain (under development)
- SAREF4EHAW: eHealth/Ageing-well domain
- SAREF4WEAR: Wearables domain
- SAREF4WATR: Water domain

Currently, in the context of URBANAGE, two SAREF extensions have been identified as relevant for the project's purposes:

- SAREF4BLDG
- SAREF4EHAW

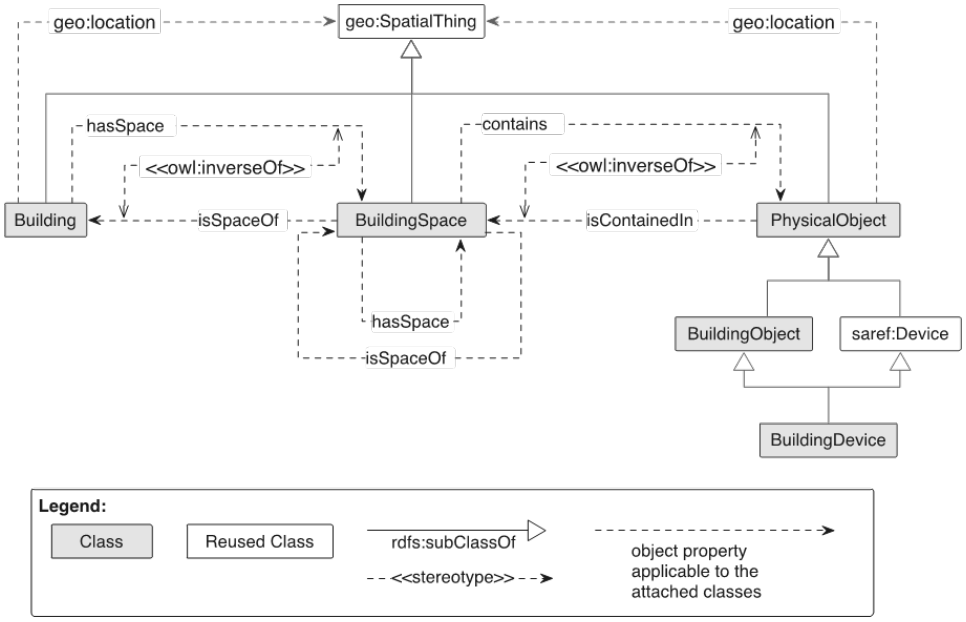
SAREF4BLDG [11] extends classes, properties and data types provided by SAREF ontology to specifically enable a representation for building domain and their physical objects. For example, the classes *s4bldg:Building*, *s4bldg:BuildingSpace* and *s4bldg:PhysicalObject* have been declared as subclasses of the class *geo:SpatialThing* in order to reuse the conceptualization for locations already proposed by the geo ontology.

[7] Image from SAREF ontology web page https://saref.etsi.org/core/v3.1.1/

The concepts *s4bldg:Building* and *s4bldg:BuildingSpace* are related to each other by means of the properties *s4bldg:hasSpace* and *s4bldg:isSpaceOf*.

The relationship between building spaces and devices and building objects has also been transferred and generalized from the core SAREF ontology. Regarding this aspect, a *s4bldg:BuildingSpace* can contain individuals belonging to the class *s4bldg:PhysicalObject*. This relation typology can be represented by the specific property *s4bldg:contains*. Moreover, a class representing *s4bldg:BuildingDevice* is defined as a subclass of both *saref:Device* and *s4bldg:BuildingObject* classes, then using classes and properties from the base and the extended ontology for building domain, as reported in Figure 10.

Figure 10: General overview of the top levels of the SAREF4BLDG extension[8]



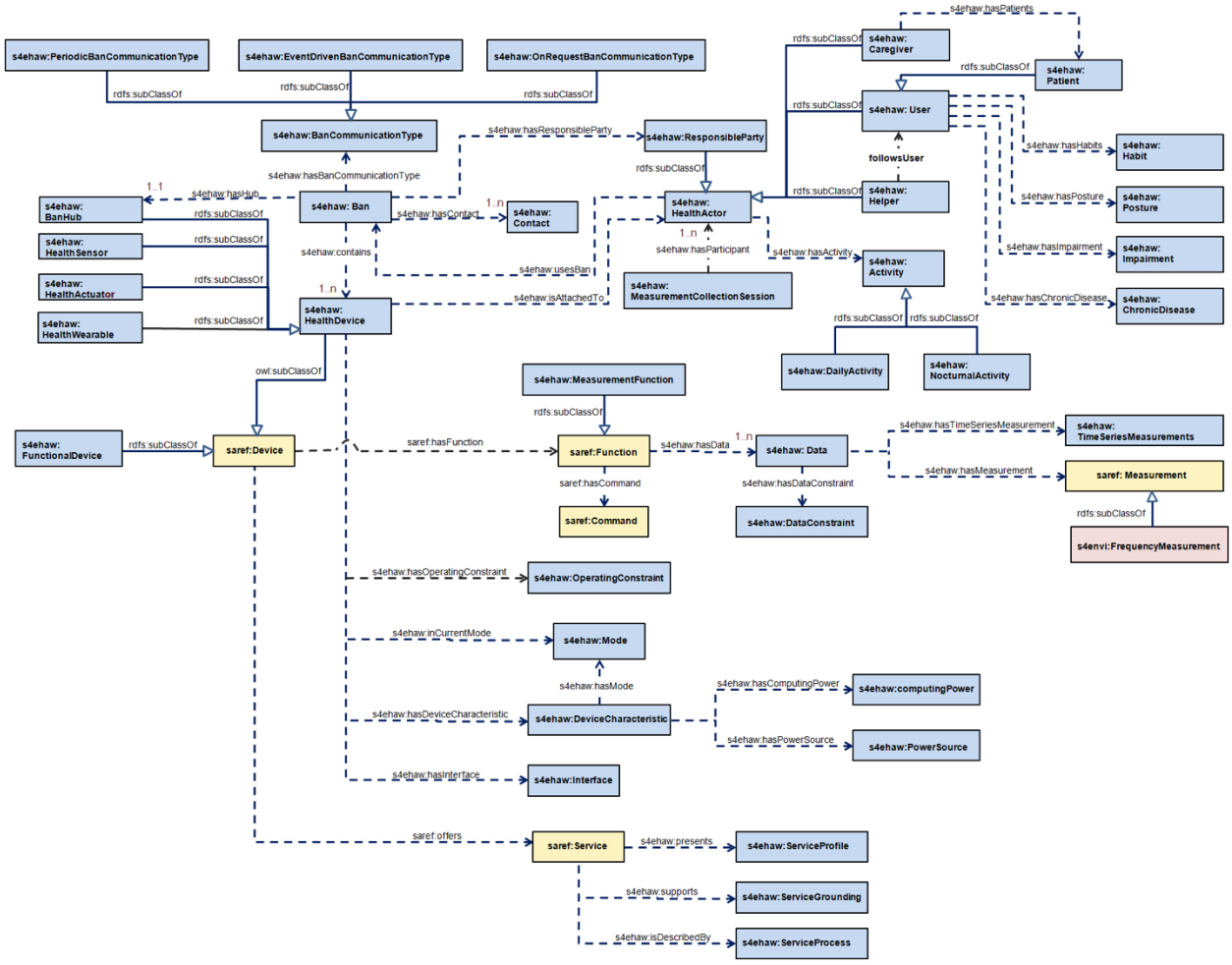
In the context of URBANAGE, diverse devices, sensors, buildings typology (such as public buildings and offices) could be potentially taken into account. For example, typical buildings could be postal offices, medical centre, pharmacies, recreation centre etc. In this sense, SAREF4BLDG offers elements and concepts that are interesting for URBANAGE project.

SAREF4EHAW [12] stands for SAREF for the eHealth/Ageing-well. It is an extension of the SAREF ontology that has been specified and formalised by investigating "eHealth and Ageing-well" domain related resources.

SAREF4EHAW, by extending SAREF ontology for the eHealth and Ageing-well vertical, make uses of following concepts: system actors, health devices, contacts that link system actors to health devices, wearable devices, and classes mainly used for collecting, aggregating and relaying patient/user vital parameters and measurements.

[8] Image from SAREF4BLDG web page https://saref.etsi.org/saref4bldg/v1.1.2/

Figure 11: High level view of the envisioned semantic model for SAREF4HAW ontology[9]



The semantic model for SAREF4EHAW ontology makes use of diverse "properties" oriented to health domain: "users" that can have "habits" for example. URBANAGE can leverage this ontology, for instance to represent "habits" of older people extending this concept also to mobility or “pedestrian movements” in general (such as movements around specific point of interest). Strictly vertical properties related to health and vital parameter are not taken into account at the moment.

[9] Image from SAREF4EHAW web page https://saref.etsi.org/saref4ehaw/v1.1.1/

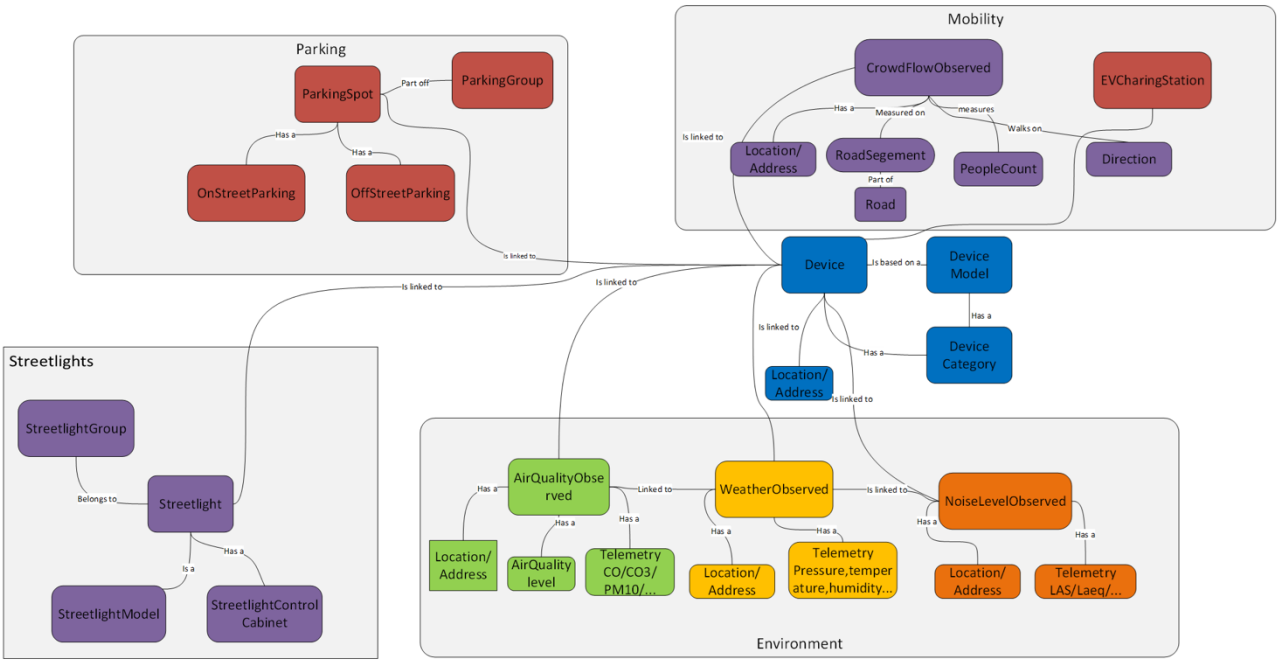
3.3 Data models

Data Models play a crucial role because they define the harmonised representation formats and semantics that will be used by applications both to consume and to publish data.

This section provides an overview of some NGSI-LD data models identified as possible candidate for URBANAGE. These data models are mainly related to the Smart Data Models Initiative[10]. This initiative is currently supported by the FIWARE Foundation with the contribution of other organizations. Smart Data Models is a collaborative initiative to provide multisector agile standardized free and open-licensed data models based on actual use cases and open standards. The provided data models are designed to keep relations between them leveraging both the ontologies on which they are based and the NGSI-LD capabilities.

Figure 12 depicts a set of NGSI-LD compliant data models related to domains and provided by the Smart Data Models initiative. The picture also reports the relation among them.

Figure 12: Example of relations between data models belonging to different domains[11]



The next sections offer a summary of currently identified data models useful for project purposes The full list of data models is available on GitHub[12] with their relative documentation and JSON schemas.

[10] https://smartdatamodels.org/
[11] Image from Smart Cities Ontology for Digital Twins web page https://techcommunity.microsoft.com/t5/internet-of-things-blog/smart-cities-ontology-for-digital-twins/ba-p/2166585
[12] https://github.com/smart-data-models

3.3.1 SmartCities – UrbanMobility[13]

This is a collection of data models correlated to urban mobility. It includes 16 data models that transpose most of the information provided by the Google format General Transit Feed Specification (GTFS) [13] that defines a common format for public transportation schedules and associated to geographic information into the NGSI-LDLD information model, such as the route of a bus, its stops, the frequency, etc.
In particular, three data models are interesting for the purposes of the project, since they make use of the field "wheelChairAccessible" to indicates wheelchair accessibility:

- **GtfsTrip** - Identify a GTFS trip with data about trip (direction, shortName, destination, etc)
- **GtfsStop** - Identify a GTFS stop (stop, station or station entrance) and data (stop name, URL of a web page about the location, location type, etc)
- **GtfsStation** - Identify a GTFS Station and data about station (name, location, address, type, relation with GTFS stop, etc)

The field "wheelChairAccessible" can have the following values:

- 0 - No accessibility info
- 1 - One rider in a wheelchair can be accommodated
- 2 - No riders in wheelchair can be accommodated

3.3.2 SmartCities – Building[14]

The SmartCities Building is a collection of data models related to building information. This collection is composed of three main data models; the more relevant ones for building and for building operation are:

- **Building** - Information on a given Building.
- **Building Operation** - Information on a given Building Operation.

More specifically, while the Building data model provides information on a specific building, the data model Building Operation contains a harmonised description of a generic operation, related to a smart building.

A partial list of more relevant properties of Building data model with a brief description is reported below:

- category: Category of the building.
- openingHours: Opening hours of this building.
- address: The mailing address.
- description: A description of this item.
- type: NGSI Entity type.
- refMap: Reference to the map containing the building.

[13] https://github.com/smart-data-models/dataModel.UrbanMobility
[14] https://github.com/smart-data-models/dataModel.Building

Similarly, regarding the Building Operation data model a list of more useful properties is reported below:

- operationType: Type of the operation on the building.
- refBuilding: Building reference where the operation is performed.
- refOperator: Reference to the Operator doing the operation on the building.

3.3.3 SmartCities – QueueMonitor[15]

The QueueMonitor data model was originally defined by the SynchroniCity[16] project. This data model allows to monitor queues on a daily run. Indeed, it describes a single queue line for a single service (for instance provided by an office located in a building).

The QueueMonitor data model makes use of different "properties"; the most relevant are:

- operationType: Type of the operation on the building.
- refBuilding: Building reference where the operation is performed.
- refOperator: Reference to the Operator doing the operation on the building.

URBANAGE could leverage this data model to manage information related to queues for specific buildings, such as post offices in a public or private structure.

3.3.4 SmartCities – PointOfInterest[17]

This collection includes data models to represent points of interest; three data models are for specific types of points of interest (Museum, Store, and Beach). the fourth data model (PointOfInterest) is more general and allows to represent a generic point of interest. All these data models could be useful to represent points of interest for older people. A full taxonomy defining the categories of points of interests is available on GitHub[18].

- **Museum**: It is a data model contain model for a museum in a city. Its more relevant properties are:
  - buildingType: Type of building that hosts the museum.
  - facilities: Describes different facilities offered by this museum. (For example, enumeration values are: elevator, cafeteria, shop, auditory, conferenceRoom etc)
  - featuredArtist: Main featured artist(s) at this museum
  - museumType: Type of museum according to the exhibited content. (For example, enumeration values are: appliedArts, scienceAndTechnology, fineArts, music, etc)
  - name: The name of this item/museum.
- **Store**: It is a data model contain models stores/shops in a city. Its more relevant properties are:
  - category: Category of the store.
  - description: A description of this item

[15] https://smart-data-models.github.io/dataModel.QueueManagement/QueueMonitor/doc/spec.md
[16] Synchronicity Project - https://cordis.europa.eu/project/id/732240/en
[17] https://github.com/smart-data-models/dataModel.PointOfInterest
[18] PointOfInterest Taxonomy - https://raw.githubusercontent.com/Factual/places/master/categories/factual_taxonomy.json

- email: The email address of this store.
- telephone: The telephone number of this store.
- type: NGSI Entity type. It has to be Store.
- name: The name of this item.

- **Beach**: It is a data model contain model for a beach. Its more relevant properties are:
  - accessType: how it is possible to access the beach; possible values are: privateVehicle, boat, onFoot, publicTransport.
  - beachType: generic characterisation of the beach; examples of possible values are: whiteSand, urban, isolated, calmWaters, strongWaves, windy, blackSand, etc.
  - facilities: facilities offered by a beach; exaples of possible values are: promenade, showers, cleaningServices, lifeGuard, sunshadeRental, sunLoungerRental, waterCraftRental, toilets, touristOffice, litterBins, telephone, accessforDisabled, etc.
  - occupationRate: typical occupation rate of this beach; possible values are: low, medium, high, none.
  - peopleOccupancy: current amount of people
  - areaServed: the geographic area served by the beach.
- **PointOfInterest**: It is a data model contains a harmonised description of a generic Point of Interest. Its more relevant properties are:
  - category: Category of this point of interest.
  - name: The name of this item.
  - type: NGSI Entity type that has to be PointOfInterest.
  - areaServed: the geographic area where the point of interest is located (and served by it).

3.3.5 SmartCities – ParksAndGardens[19]

This data model collection is composed of three diverse data models as follow described:

- **Garden**. A garden is a distinguishable planned space, usually outdoors, set aside for the display, cultivation, and enjoyment of plants and other forms of nature.
- **GreenspaceRecord**. This entity contains a harmonised description of the conditions recorded on a particular area or point inside a greenspace (for example flower bed, garden, etc.).
- **FlowerBed**. A garden plot in which flowers (or other plants) are grown. Usually, you will find flower beds in parks, gardens, pedestrian areas or at big highway interchanges.

[19] https://github.com/smart-data-models/dataModel.ParksAndGardens

3.3.6 SmartCities – Transportation[20]

This is a collection of data models related to transportation. It comprises 22data models; the more relevant for URBANAGE are:

- **TransportStation** - The data model is a general description of urban stations (Metro, Bus, Tram, Heliport, etc.) according to the GTFS standard.
- **RestrictedTrafficArea** - An area of a city in which the traffic generated by cars or any other kind of vehicles is subjected to limitation.
- **BikeHireDockingStation** - Bike Hire Docking Station.

[20] https://github.com/smart-data-models/dataModel.Transportation

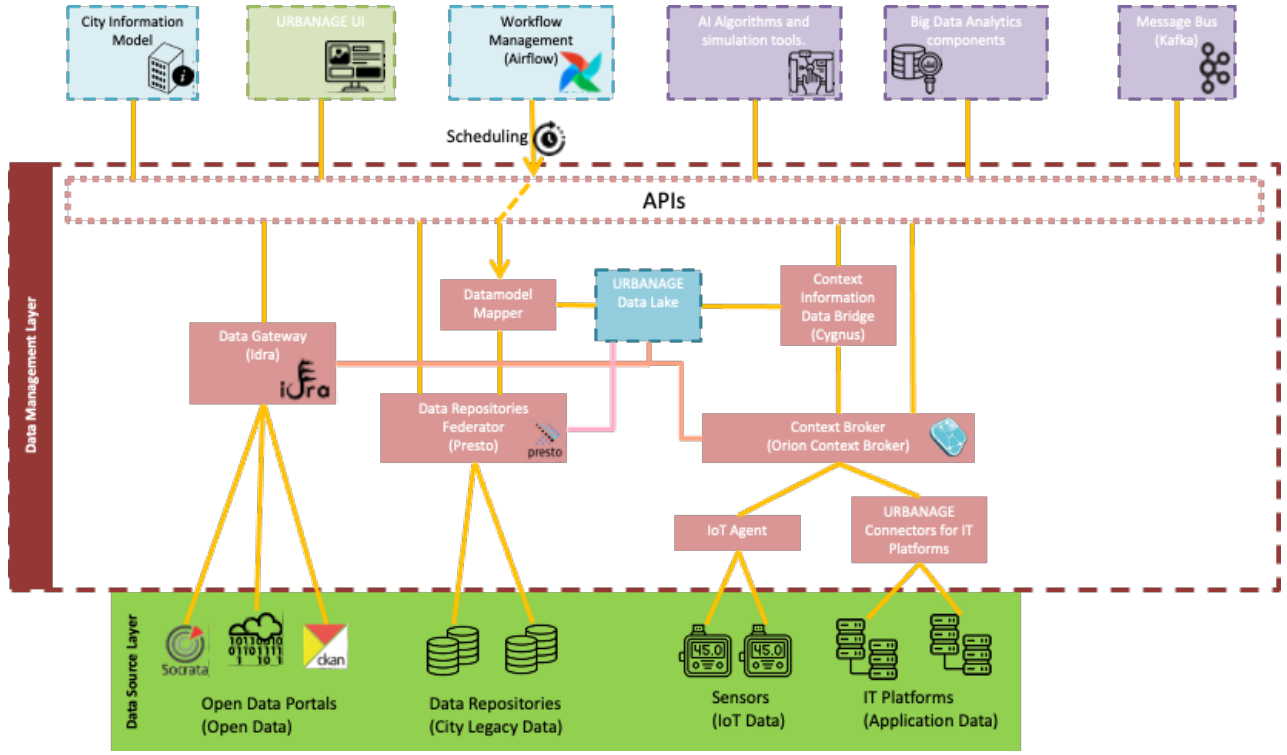
4 Data Management Layer

The Data Management Layer offers the functionalities to access, collect, aggregate and harmonise (leveraging common ontologies and NGSI-LD standard) both static and real-time data coming from heterogeneous sources, such as databases, Open Data Management Systems (e.g. CKAN, Socrata, etc.), existing IT platforms (e.g. legacy systems) and sensors.
The data collected and harmonised by the Data Management Layer belong to three categories: static data coming from repositories (such as databases); real-time coming from IoT devices or existing IT platforms; metadata of datasets managed by Open Data Management Systems.
The managed data is so exposed and made available to the other components of the URBANAGE platform (e.g. to apply analytics).
Under this perspective, the Data Management Layer acts as a bridge between the data sources and the high-level components of the URBANAGE platform, enabling the latter to access the data they need through a unified data API and model. These components are mainly represented by the “Big Data Analytics”, the “AI Algorithms and simulation tools” the “City Information Model” and the “URBANAGE UI”; in addition, other two components interact with the Data Management Layer: the “Workflow Management” and the “Message Bus”.

This section, first introduce the logical components of the Data Management Layer, the relations among them and technological tools to build them. Then describes the general processes followed by the Data Management Layer for the data collection, harmonisation and aggregation processes. Finally, an overview of the APIs exposed by the Data Management Layer is provided.

Figure 13 depicts the general overview of the overall Data Management Layer, the relations among its internal components and with the other components of the URBANAGE Platform that can interact with it A brief description of the main components of the URBANAGE Platform that interacts with the DML is reported in *Annex 2 – Main components of URBANAGE Platform interacting with the Data Management Layer*.

Figure 13: Overview of the URBANAGE Data Management Layer[21]



4.1 Components of the Data Management Layer

The Data Management Layer (DML) is composed of seven components; these are the Data Gateway, the Data Repositories Federator, and the Context Broker, which are also the core components of the DML, the Datamodel Mapper, the Context Information Data Bridge, the IoT Agents, and the URBANAGE Connectors for IT Platforms.

Figure 13 depicts within the boundaries of the DML also the URBANAGE Data Lake, that is the part of the Big Data Analytics components devoted to the storage of the data feeding the (big data) analysis. The URBANAGE Data Lake is depicted within the DML to better clarify the connections with the components of the DML itself with it. More details about the URBANAGE Data Lake are available in the deliverable "D3.5 Big Data Analytics components (Initial)"

The **Data Gateway** (based on Idra[22]) provides the functionalities to search and discover the datasets managed both by the URBANAGE platform, as well as external systems, such as Open Data Management Systems (e.g.

[21] The purpose of lines with different colours in the figure is to better represent the connections among the modules of the Data Management Layer

[22] Idra provides a unique access point to search and discover datasets managed by heterogeneous sources. It uniforms representation of the metadata of collected datasets according to DCAT-AP and provides a set of RESTful APIs to be used by third party applications. More details are available at https://github.com/OPSILab/Idra

CKAN, Socrata, etc.). Data Gateway works as a unique access point to search and discover open datasets. To this aim, it harmonises the metadata for datasets provided also by external platform according to DCAT-AP format [14]. It provides a collection of RESTful APIs to interact with third-party applications and other URBANAGE components. Moreover, it is also able to federate generic open data portals that don't expose APIs by using web scraping functionality[23]. The Data Gateway offers a user interface, that will be exposed through the general URBANAGE UI. This user interface provides both *administrative* functionalities useful for example to federate open data portals and also *end-user* functionalities to create a graphical and reusable visualization of open data.

The **Data Repositories Federator** (based on Presto[24]) allows to perform SQL queries against heterogeneous data repositories, both internally within the URBANAGE platform (i.e. URBANAGE Data Lake) and externally, towards data repositories owned by third parties (such us legacy data repositories of the city). The Data Repository Federator interacts with a data repository through embedded connectors; since the Data Repository Federator is based on Presto, it includes connectors to support different technologies, e.g. Cassandra, Hive, Kafka, MongoDB, MySQL, PostgreSQL, etc. Indeed, it allows to interact with different databases or even proprietary data stores combining potentially in a single query data coming from multiple sources and diverse technologies, working as a distributed SQL query engine for running interactive analytic queries against data sources of big sizes.

The Data Repository Federator is also connected to the **Datamodel Mapper**, a tool able to translate structured input data into NGSI compliant entities, that are then stored into the URBANAGE Data Lake for further analysis; currently, one of the main candidates for the realisation of the Big Data Analytic Storage is Min.io[25]. The Datamodel Mapper is able to convert several file types (e.g. JSON, GeoJSON, CSV) according to different data models. This tool uses a mapping file, which is a well-formed JSON file, in order to know how to map each source field of the parsed row/object in the destination fields. The file in input can contain either rows, JSON objects or GeoJSON features, each of them representing an object to be mapped to an NGSI entity, according to the selected data model passed to the tool as an input parameter.

Leveraging the **Workflow Management**, it is possible to schedule recurrent and pre-configured query against federated data repositories to collect and store updated data. The Workflow Management is based on Apache Airflow[26]. The Workflow Management allows to schedule a workflow by time or events, and it offers an

[23] By defining a configuration file named "Sitemap", Idra is instructed on how to navigate the portal and collect the needed information, mapping the metadata in the DCAT-AP format. Idra also offers a plugin for the Chrome browser to facilitate the definition of the Sitemap. More information is available on the dedicated section of Idra user manual - https://idra.readthedocs.io/en/latest/admin/scraping/

[24] Presto is a distributed SQL query engine able to perform queries over multiple heterogeneous data sources. More details are available at https://prestodb.io/docs/current/index.html

[25] An open-source tool offering object storage capabilities compatible with S3 (Simple Storage Service) APIs https://min.io/

[26] An open-source tool written in Python that uses this language to define workflows following, behind the woods, the principle called “workflow as a code” https://airflow.apache.org/

administrative user interface to monitor workflows statutes, outputs, configuration parameters and logs. Airflow offers a wide range of integration options and protocols to interact with.

The **Context Broker** (based on the Orion Context Broker[27]) manages the life cycle of context information and the dispatching of real-time information following a publish-subscribe approach. The context information is represented in the form of NGSI-LD entities. The Context Broker receives data from the **IoT Agents** and from the **URBANAGE Connectors for IT Platforms**.

- The IoT Agents are FIWARE Generic Enablers that facilitate the connection of IoT devices to gather context information from them; they can also trigger actuations in response to context updates. The IoT Agents support different protocols, such as Lightweight M2M[28], UltraLight[29], Sigfox,[30] and LoRaWAN[31].
- URBANAGE Connectors for IT Platforms are more complex components mainly devoted to allowing a bidirectional communication between the URBANAGE Platform and legacy IT systems. They consist of a tailored implementation of a set of specifications that exposes a uniform interface enabling he connection to the URBANAGE Platform (Section 4.1.1 provides more details on the URBANAGE Connectors for IT Platforms).

Finally, the Context Broker is connected to the **Context Information Data Bridge** (CIDB) component (based on FIWARE Cygnus[32]); this component acts as a bridge between the Context Broker and the URBANAGE Data Lake. CIDB persists the NGSI-LD entities managed by the Context Broker creating a historical view of such information and of their related attributes. CIDB supports different technologies and tools. Among them HDFS, MySQL, MongoDB, Kafka, PostgreSQL, Carto (a database specialized in geolocated data), Elasticsearch (a distributed search and analytics engine).

To retrieve the NGSI-LD entities to be stored, CIDB leverages the publish-subscribe capabilities of the Context Broker. To this aim, a subscription must be created on the Context Broker; the subscription must specify the entities to be notified to CIDB when an update occurs and the attributes of interest. So, the notified attributes of the NGSI-LD entities are persisted.

CIDB plays the role of a connector between Context Broker and different kinds of tool. Indeed, it is the bridge between the Context broker and the URBANAGE Data Lake, but also with the **Message Bus**.

[27] The Orion Context Broker is one of the building blocks promoted by the CEF Programme and the main Generic Enabler of the FIWARE platform. It offers an implementation of both NGSI-LD and NGSIv2 APIs. More details are available at https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/Orion+Context+Broker and at https://github.com/FIWARE/context.Orion-LD

[28] https://fiware-iotagent-lwm2m.readthedocs.io/en/latest/

[29] https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual/index.html

[30] https://iotagent-sigfox.readthedocs.io/en/latest/

[31] https://fiware-lorawan.readthedocs.io/en/latest/

[32] A connector in charge of persisting/transmitting context information to a configured storage/tool. In turn, Cygnus is a customisation of Apache Flume https://fiware-cygnus.readthedocs.io/en/latest/

Table 2 briefly summarises the baseline technical tools of the Data Management Layer.

Table 2: Data Management Layer baseline technological tools[33]

Name	Brief Description	Role	License
Idra	It is as unique point of access to search and discover datasets from URBANAGE platform and form external Open Data Management Systems.	Data Gateway	AGPL 3.0
Presto	It is a distributed SQL query engine for running interactive queries against data sources from heterogeneous data repositories.	Data Repository Federator	Apache 2.0
Apache Airflow[34]	It is a workflow orchestrator that let to schedule pipeline by time or events leveraging a wide range of integration options providing a UI to monitor and check executions.	Workflow Management	Apache 2.0
Data Model Mapper	It is a tool able to map and translate structured input data into a NGSI entities according a selected data model.	Data Model Mapper	AGPL 3.0
Orion Context Broker	It manages the life cycle of context information dispatching real-time information received from IoT Agents following a publish-subscribe approach.	Context Broker	AGPL 3.0
IoT Agents	IoT Agents support different protocols (such as Lightweight M2M, UltraLight, Sigfox and LoRaWAN) to gather context information provided by IoT devices through IoT connectors.	IoT connectors	AGPL 3.0
Cygnus	It persists NGSI-LD entities managed by the Context Broker creating an historical view of context information and of their related attributes.	Context Information Data Bridge	AGPL 3.0

[33] For development purposes of the URBANAGE Platform, the project established a CI/CD (Continuous Integration and Deployment) process, that includes among the tools to be used, a code repository (i.e. GitLab). The code repository collects the prototypes of the main components (e.g. baseline tools, libraries, datasets, etc.) constituting the Data Management Layer and their future developments. The deliverable "D5.2 Initial Platform Prototype" provides details about the CI/CD process and the code repository.

[34] Even if the Workflow Management (the components based on Airflow) is not part of the DML, it is important to report also its relative baseline tool to provide a more complete view.

4.1.1 URBANAGE Connector for IT Platform

The URBANAGE Connector for IT Platforms is composed of two parts: one part on the side of the URBANAGE Platform and another one on the side of the connected IT Platform. The latter, from a conceptual point of view, is composed of different logic layers:

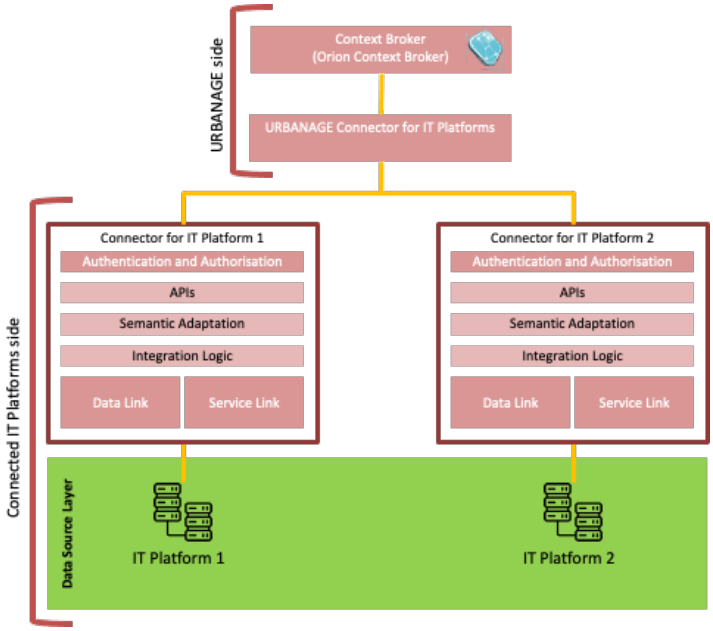
- **Authentication and Authorisation** layer protects the connector against external malicious attacks and avoids leakages of data.
- **APIs** layer implements a set of REST APIs to allow the concrete interaction between the Data Management Layer and the connector. APIs provides a uniform set of methods, offering a common way to access the services and the data exposed by the connected IT platform. The concrete implementation of these APIs depends on the specific integration needs required for the underlying IT Platform. The APIs of the URBANAGE Connector for IT Platform (both on URBANAGE and IT Platform sides) follow the APIs of IDS (International Data Spaces) connector provided by the TRUE (TRUsted Engineering) Connector[35]. This implementation uses both HTTP and HTTPS protocols and exposes three endpoints:
  - ***/proxy*** *to receive data incoming request*
  - ***/data*** *to receive data from a sender*
  - ***/about/version*** *returns business logic information*

  Detailed information is available on TRUE Connector official documentation.
- **Semantic Adaptation** layer performs the "translation" of the exchanged data according to the data models managed by the URBANAGE Platform (i.e. NGSI-LD based).
- **Integration Logic** layer performs the procedures needed to allow the correct integration of the connector with the specific (legacy) IT Platform. The procedures and operations needed to integrate the connector with the specific IT Platform will depend on the policies, technical characteristics, restrictions, requirements, etc. of the latter. The Integration Logic layer leverages other two logical layers that allow it to access the data and to interact with the services of the specific IT Platform (the Data Link and the Service Link layers). The Integration Logic layer will allow performing anonymisation of the data if needed.
- **Data Link** layer enables the Integration Logic layer to access the data managed by the specific IT Platform; the way the Data link layer will accesses data will depend on the policies, technical restrictions, requirement, etc. of the specific IT Platform.
- **Service Link** layer enables the Integration Logic layer to interact with services exposed by the specific IT Platform; the way the Data Link layer interact with those services will depend on the policies, technical restrictions, requirement, etc. of the specific IT Platform.

Figure 14 depicts the general diagram of a connector for IT platform, highlighting its main components.

[35] https://github.com/Engineering-Research-and-Development/true-connector

Figure 14: Schematic representation of Connector for IT Platform



Suggested tools

This section briefly reports a list of suggested tools for the implementation of a generic URBANAGE Connector of IT Platform.

Concerning the Authentication and Authorisation capabilities, the recommended tool is **Keycloak**[36], also, in continuity with the overall technical approach of the URBANAGE Platform (detailed in the deliverable "D5.1 System Architecture & Implementation Plan"). Keycloak is an open-source Identity and Access management tool that supports standard protocols, such as Oauth2[37].

Semantic adaptation can be supported by a tool already included in the DML. The **Datamodel Mapper** can be adopted also in this case to translate data coming from the IT Platform.

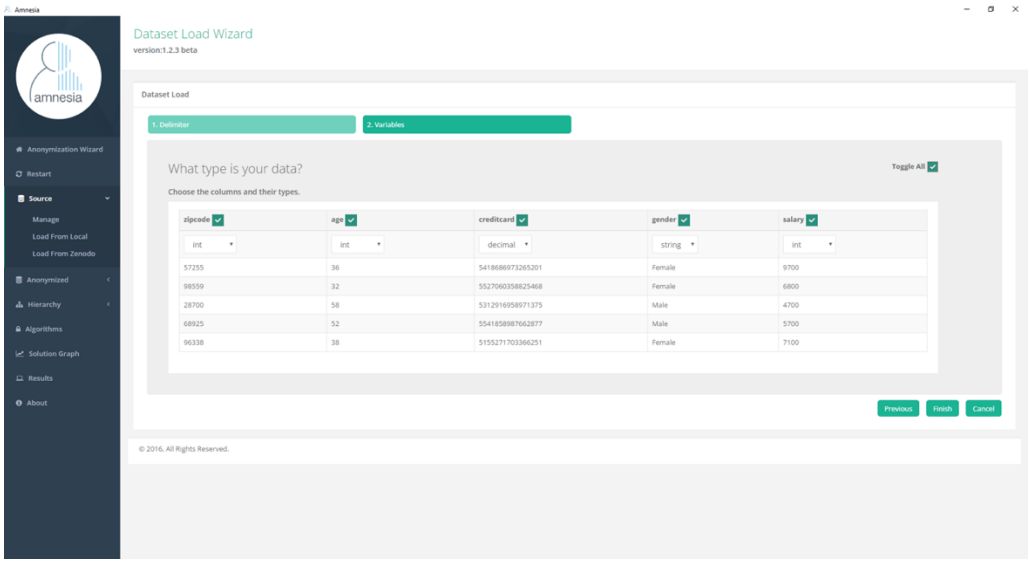
Finally, the Integration Logic, the Data Link and the Service Link are strictly related to the technical characteristics of the IT Platform (e.g. Exposed APIs, databases, framework, programming languages, etc.) and in many cases they should be implemented from scratch. However, some tools can facilitate this process; tools and frameworks that support ETL (Extract Transform Load) operations can offer a significant advantage for the implementation of these three layers of a connector.

As for the Authentication and Authorisation layer, in continuity with the overall technical approach of the URBANAGE Platform, the suggested tool for the implementation of the Integration Logic, the Data Link and the Service Link is **Airflow**. In the case of anonymisation needs, tools such as **Amnesia**[38] can be usedFigure 15); Amnesia is a tool promoted by Open AIRE[39] that offers a flexible solution for the anonymization process and leverages algorithms that take advantage of modern hardware architectures based on multiple cores.

[36] https://www.keycloak.org/
[37] https://oauth.net/2/
[38] https://amnesia.openaire.eu/index.html
[39] https://www.openaire.eu/

Moreover, Amnesia simplifies the definition of anonymization and guides the users by its user interface to the outcomes of the anonymization process.

Figure 15: User interface of Amnesia



4.2 Data collection, harmonisation and aggregation processes

This section provides a high-level overview of the processes applied by the DML to manage both static and real-time data (collection, harmonisation, aggregation and storage).

4.2.1 Collection and harmonisation of static data

This section describes the high-level process for the collection and harmonisation of static data coming from generic data repositories; the sample scenario described here considers two data repositories.

The Workflow Management component initiates the flow. It triggers the Datamodel Mapper providing different information:

1) the query to be performed against the Data Repository Federator
2) the NGSI-LD data model to be followed for the harmonisation of the data fetched by the Data Repository Federator,
3) the mapping instructions to translate the fetched data into NGSI-LD entities compliant with the indicated data model.

Then the Datamodel Mapper interacts with the Data Repository Federator to fetch the data; the latter

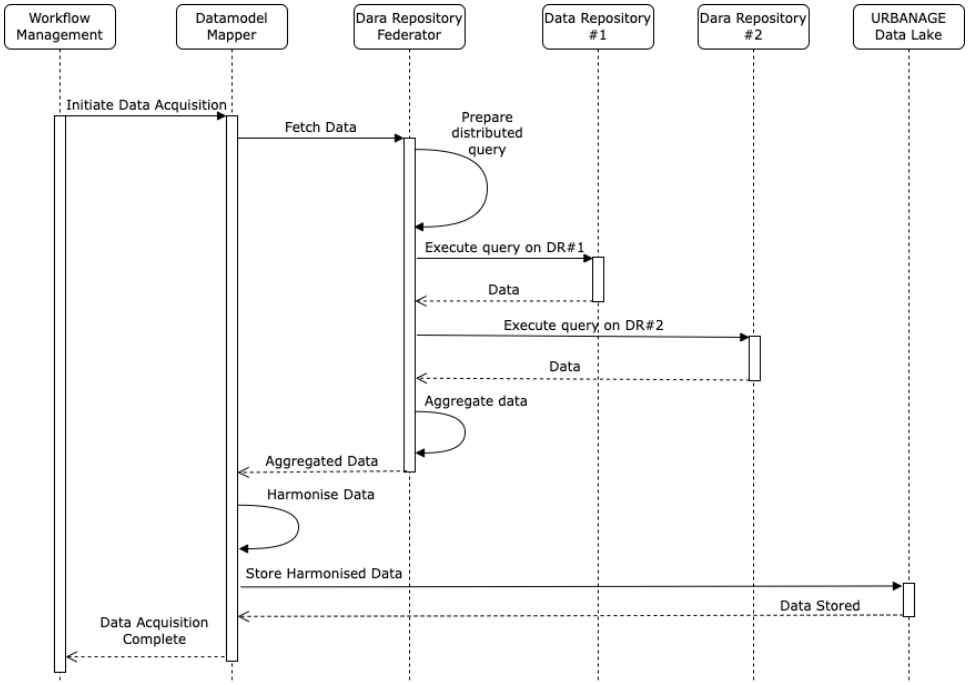
1) prepares the queries to be executed on the two federated data repositories
2) perform the two queries against them, getting in return the corresponding data.

Once the Data Repositories Federator obtains the whole sets of data, it aggregates and integrates them according to the received instructions.

Then the data is provided to the Datamodel Mapper, which performs the harmonisation of the received data according to the mapping instructions and the NGSI-LD data model.

Once the harmonisation is completed, the Datamodel Mapper stores the harmonised data (the NGSI-LD entities) on the URBANAGE Data Lake.

Figure 16: Sequence diagram - Collection and harmonisation of static data



4.2.2 Collection and harmonisation of real-time data

This section describes the high-level process for the collection and harmonisation of real-time data coming from a generic sensor.

The flow is initiated by the Sensor (e.g. a physical device) that pushes an update to the corresponding IoT Agent (a software component associated with the sensor acting as a bridge between it and the rest of the DML). The IoT Agent treat the received data and create a corresponding NGSI-LD entity representing it; this step corresponds to the harmonisation process. The specific NGSI-LD is created according to the specific type of data and is configured during the set-up of the IoT Agent.

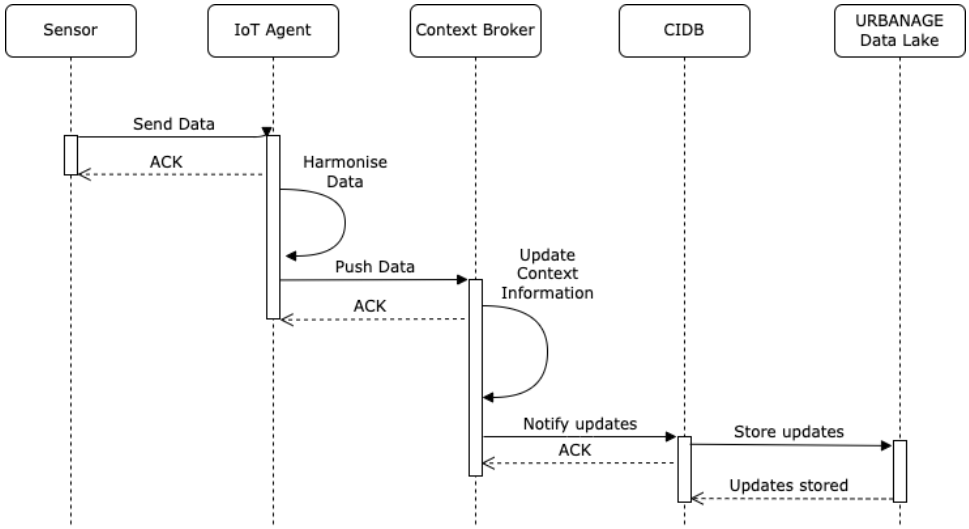
Once ready, the IoT Agent sends the NGSI-LD entity containing the data provided by the sensors to the Context Broker that updates the managed context information.

Once the Context Broker receives the data from the IoT Agent, two situations can occur.

1) The data present a new sensor (with the corresponding measure); for instance, a temperature sensor. In this case, the Context Broker instantiates the corresponding entities among the managed context information
2) The data represent an update associated with a sensor; for instance, the temperature detected by the sensor. In this case, the Context Broker updates the corresponding entity already existing among the managed content information.

In both cases, the updates (instantiation of a new entity or its update) are notified to the CIDB (Context Information Data Bridge) that is in charge of persisting them into the URBANAGE Data Lake, allowing so the collection of historical data that can be used for further analysis.

Figure 17: Sequence diagram - Collection and harmonisation of real-time data

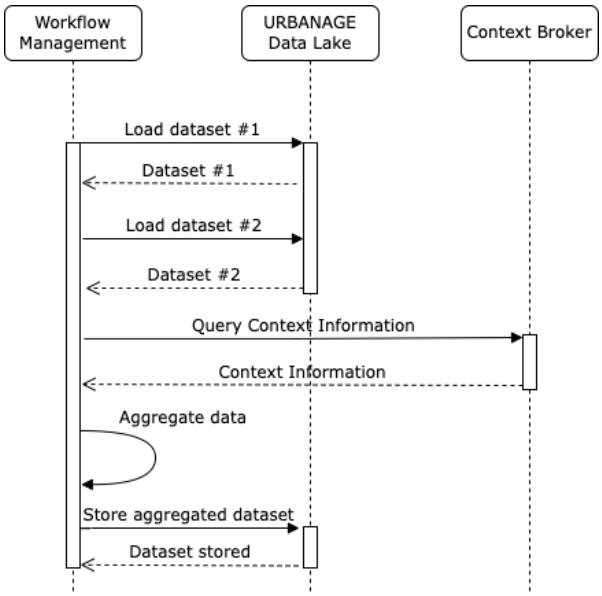


4.2.3 Data Aggregation and Integration

As described in section 4.2.1, the process for the collection and harmonisation of static data foresees also the possibility to aggregate data coming from scattered (static) data sources (e.g. databases). Static data retrieved by the Data Repository Federator can be aggregated at the time of collection and stored into the URBANAGE Data Lake already aggregated (after being harmonised by the Datamodel Mapper). However, if it is necessary to perform further data aggregation, it is possible to leverage the Workflow Manager to perform this operation.

As reported in section 4.1, the Workflow Manager is based on Apache Airflow; this tool can execute Python scripts. So, it is possible to programmatically design and code the needed steps to perform the further data aggregation operation of the information managed within the URBANAGE Data Lake, combining them also with the Context Information managed by the Context Broker. Since Airflow offers the possibility to connect with S3 APIs, it could directly interact with the URBANAGE Data Lake; alternatively, the Data Repository Federator can act as an interoperability layer between the Workflow Manager and the URBANAGE Data Lake. Figure 18 briefly depicts this process.

Figure 18: Sequence diagram – Sample data aggregation process



Since the purpose of data aggregation is to produce datasets ready to be analysed or visualised, the specific data aggregation operations that will be performed by URBANAGE Platform (e.g. through the Workflow Manager or by the usage of the Data Repository Federator) will rely on the specific use cases to be implemented and on the available datasets. Deliverable D5.2 Initial Platform Prototypes provide more information about the approaches for data integration.

4.3 Overview of Data Access APIs

This section provides an overview of the APIs exposed by the Data Management Layer to allow the other components of the URBANAGE architecture to access the managed data. The main components of the Data management Layer involved in interaction with the other parts of the URBANAGE architecture are the Data Gateway, the Data Repository Federator, the Context Broker, and Context Information Data Bridge.

Data Gateway

Table 3 provides an overview of the more relevant APIs exposed by the Data Gateway. In particular, this component (based on Idra) provides different APIs typologies: end-user APIs (with “client” path), federation APIs (with “odf” path) and administration APIs (with “administration” path). All resources/URIs have the following root path: *{IdraApiRoot}/Idra/api/v1/*. For more information, please refer to the Idra official documentation [15].

Table 3: APIs exposed by Data Gateway

Resource Name	Resource URI	HTTP Method	Description
Datasets Search	/odf/odms/search	POST	This API is in charge of performing the federated search on all the dataset metadata belonging to federated ODMS catalogues.
Datasets Summary	/odf/odms/datasets/info	GET	This API is in charge of retrieving the summary list of all available datasets providing the identifiers and the release Date and update Date.
Retrieve Single Dataset	/odf/odms/datasets/{datasetID}	GET	This API is in charge of retrieving the dataset identified by the requested *datasetID*.
End User Catalogues Resources	/client/catalogues	GET	This API is in charge of retrieving the metadata of the federated ODMS catalogues.
Single Catalogue APIs	/client/catalogues/{nodeId}	GET	This API is in charge of retrieving a specific federated ODMS catalogue by its *nodeId*.
Catalogues Info	/client/cataloguesInfo	GET	This API is in charge of retrieving a subset of the metadata of the federated ODMS catalogues that allow the search operation; specifically, the following metadata is retrieved for each catalogue: id, the federation level and the name.
Metadata Search	/client/search	GET	This API is in charge of performing the federated search on all the dataset metadata belonging to federated ODMS catalogues. It is possible to specify by filters some parameters about search: if search is multilingual, or if search is live or cached etc.
Catalogues Resources	/administration/catalogues	GET	This API is in charge of retrieving the details of all federated ODMS catalogues.
		POST	This API is in charge of adding a new ODMS catalogue to the federation.
Single Catalogue APIs	/administration/catalogues /{nodeId}	GET	This API is in charge of retrieving a specific federated ODMS catalogue by its *nodeId*.
		PUT	This API is in charge of updating a specific federated ODMS catalogue by its *nodeId*.
		DELETE	This API is in charge of deleting a specific federated ODMS catalogue by its *nodeId*.
Synchronize Catalogue	/administration/catalogues /{nodeId}/synchronize	POST	This API is in charge of forcing the synchronization of the Catalogue identified by the nodeId parameter.

Data Repository Federator

Table 4 provides an overview of the more relevant APIs exposed by Data Repository Federator. This component is based on Presto and its APIs can be grouped in two categories: Client APIs (to submit queries) and Worker APIs (to coordinate the query execution among different nodes). For more information, please refer to the official documentation of Presto [16].

Table 4: APIs exposed by Data Repository Federator

Resource Name	Resource URI	HTTP Method	Description
Statement (Client APIs)	/v1/statement	POST	This API is in charge of executing the query included in the POST body; it returns a JSON document containing the query results. If there are more results, the JSON document will contain a *nextUri* URL attribute.
		GET	This API is in charge of providing, by using *nextUri* attribute, the next batch of query results.
		DELETE	This API is in charge of terminating a running query by using *nextUri* attribute.
Control Plane (Worker APIs)	/v1/task/{taskId}	POST	This API is in charge of executing the query fragment specified in the POST body. The request optionally includes a set of initial splits to process.
	/v1/task/{taskId}/status	GET	This API is in charge of retrieving the *TaskStatus* JSON document describing the current execution status.
	/v1/task/{taskId}	DELETE	This API is in charge of deleting a finished task or cancels a task in-progress.

Context Broker

The Context Broker provides a consistent way to manage context information and to execute different operations like subscriptions, notifications, data gathering, etc. All resource URIs have the following root path: *{apiRoot}/{apiVersion}/*. The Context Broker implements the NGSI-LD APIs, already introduced in section 3.1.2. More information about ETSI NGSI-LD APIs is available in the dedicated web page [17].

Context Information Data Bridge

This component is based on Cygnus. Table 4 provides an overview of the more relevant APIs exposed by Cygnus. Cygnus is a connector acting as a bridge between the Context Broker and other tools (such as for persisting context information and in doing so creating an historical view of such data). Cygnus is based on Apache Flume [40]. It supports different third-party storages such as HDFS, MySQL, CKAN, MongoDB etc. Further details about exposed APIs are available in the official documentation [18].

[40] Apache Flume - https://flume.apache.org/

Table 4: APIs exposed by the Context Information Data Bridge

Resource Name	Resource URI	HTTP Method	Description
Subscription	/v1/subscriptions&ngsi_version=2	GET	This API is in charge of retrieving the existent subscriptions, given the NGSI version (2 in this case). If the query parameter "subscription_id" is provided, this API retrieve only the subscription corresponding to the provided *subscription id*.
	/v1/subscriptions&ngsi_version=2	POST	This API is in charge of creating a new subscription.
	/v1/subscriptions?ngsi_version=2 &subscription_id={subscriptionId}	DELETE	This API is in charge of deleting the subscription identified by the query parameter "subscription_id".

5 Conclusion

This document describes the initial design of the Data Management Layer (DML) of the URBANAGE Platform, including the main interactions with other relevant components of the URBANAGE Platform itself, from the perspective of the DML. Within the URBANAGE Platform, the DML acts a bridge between the high-level capabilities provided by URBANAGE (e.g. Big Data Analytics, Artificial Intelligence, the Digital Twin) and the data sources needed to feed them.
The DML allows to collect data from heterogenous data sources, such as sensors, repositories (e.g. databases), (legacy) IT Systems that can expose diverse APIs and offer data in different formats. Under this perspective, the DML offers the capabilities needed to collect, harmonise and aggregate data, making it ready to be exploited by the other layers of the URBANAGE Platform. The overall processes to collect and harmonise and aggregate data are also reported and described in this document.

To ensure data interoperability, the DML makes use of a common data format (to properly structure the collected information) and ontologies (to model and establish relations among the information itself); the adoption of NGSI-LD and SAREF ontologies and their extensions (that can be further extended) offers the opportunity to follow consolidated and well-known standards, increasing the replicability and interoperability chances.

6 References

[1] ECMA, «ECMAScript Language Specification Standard ECMA-262 3rd Edition,» Geneva, 1999.

[2] «JSON for Linking Data,» [Online]. Available: https://json-ld.org/.

[3] W3C, «JSON-LD 1.1 A JSON-based Serialization for Linked Data - W3C Recommendation 16 July 2020,» [Online]. Available: https://www.w3.org/TR/json-ld/.

[4] Open Mobile Alliance, «NGSI Context Management Version 1.0,» 2012.

[5] ETSI, «ETSI GS CIM 009 V1.5.1 Context Information Management (CIM); NGSI-LD API,» 2021-11.

[6] W3C, «Resource Description Framework (RDF),» [Online]. Available: https://www.w3.org/RDF/.

[7] W3C, «RDF Schema 1.1,» [Online]. Available: https://www.w3.org/TR/rdf-schema/.

[8] W3C, «Semantic Sensor Network Ontology,» [Online]. Available: https://www.w3.org/TR/vocab-ssn/.

[9] W3C, «SOSA Ontology,» [Online]. Available: https://www.w3.org/2015/spatial/wiki/SOSA_Ontology.

[10] ETSI, «ETSI TS 103 264 V3.1.1 SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping,» 2020-02.

[11] ESTI, «ETSI TS 103 410-3 SmartM2M; Extension to SAREF; Part 3: Building Domain,» 2020-05.

[12] ETSI, «SAREF4EHAW: an extension of SAREF for eHealth Ageing Well domain,» [Online]. Available: https://saref.etsi.org/saref4ehaw/v1.1.1/.

[13] Google, «GTFS Static Overview,» [Online]. Available: https://developers.google.com/transit/gtfs.

[14] «DCAT-AP 2.1.0,» [Online]. Available: https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/dcat-application-profile-data-portals-europe/release/210.

[15] OPSILab - Engineering Ingegneria Informatica SpA, «Idra - Open Data Federation Platform,» [Online]. Available: https://idraopendata.docs.apiary.io/#.

[16] Presto, «Presto Documentation,» [Online]. Available: https://prestodb.io/docs/current/index.html.

[17] ETSI, «ETSI ISG CIM / NGSI-LD API,» [Online]. Available: https://forge.etsi.org/swagger/ui/?url=https://forge.etsi.org/rep/NGSI-LD/NGSI-LD/raw/master/spec/updated/generated/full_api.json.

[18] FIWARE, «CYGNUS,» [Online]. Available: https://fiware-cygnus.readthedocs.io/en/latest/index.html.

[19] Open Geospatial Consortium, «CityGML,» [Online]. Available: https://www.ogc.org/standards/citygml.

[20] «IDRA - OPEN DATA FEDERATION PLATFORM,» [Online]. Available: https://idra.readthedocs.io/en/latest/.

[21] Open API Initiative, «OpenAPI Specification,» [Online]. Available: https://spec.openapis.org/oas/latest.html.

[22] Route2PA EU Project, «Datalets,» [Online]. Available: https://github.com/routetopa/spod/wiki/Datalets.

[23] SynchroniCity EU Project, «SynchroniCity - Data Model Mapper,» [Online]. Available: https://gitlab.com/synchronicity-iot/data-model-mapper/-/blob/master/README.md.

[24] FIWARE, «FIWARE-NGSI v2 Specification,» [Online]. Available: https://swagger.lab.fiware.org/?url=https://raw.githubusercontent.com/FIWARE/specifications/master/OpenAPI/ngsiv2/ngsiv2-openapi.json.

[25] FIWARE, «IoT Agents,» [Online]. Available: https://github.com/FIWARE/tutorials.IoT-Agent.

[26] FIWARE, «IOT AGENT FOR JSON,» [Online]. Available: https://fiware-iotagent-json.readthedocs.io/en/latest/.

[27] FIWARE, «IOT AGENT FOR OMA LIGHTWEIGHT M2M,» [Online]. Available: https://fiware-iotagent-lwm2m.readthedocs.io/en/latest.

[28] FIWARE, «IOT AGENT FOR ULTRALIGHT 2.0,» [Online]. Available: https://fiware-iotagent-ul.readthedocs.io/en/latest/.

[29] FIWARE, «IOT AGENT FOR LORAWAN PROTOCOL,» [Online]. Available: https://fiware-lorawan.readthedocs.io/en/latest/.

[30] «FIWARE TRUE CONNECTOR,» [Online]. Available: https://github.com/Engineering-Research-and-Development/fiware-true-connector/blob/master/README.md .

[31] «Amnesia ReST API Structure,» [Online]. Available: https://github.com/dTsitsigkos/Amnesia#rest-api-endpoints.

[32] Apache Airflow, «Apache Airflow Documentation,» [Online]. Available: https://airflow.apache.org/docs/apache-airflow/2.2.3/index.html.

7 Annex 1 – Overview of Initial datasets from pilot sites

This section provides an overview of the initial datasets identified by the three pilot sites of the project (Santander, Flanders and Helsinki) that potentially could be leveraged to feed the Data Management Layer and so the whole URBANAGE Platform. For each dataset, the following information is reported: the title, a short description, the distribution formats (e.g. CSV, JSON, etc.), the license (if this information is not available "-" is reported), and a link to a web page providing more details. The datasets here reported were identified during the process for the definition of the use cases. The datasets which will be actually imported into the Data Management Layer may differ from those here reported since they will depend on the evolution of the use case needs.

Santander pilot site

Title	Description	Format	License	Link
Stops location (ideally GIS based)	Information from Network Bus stops deployed in the city of Santander and their geographical position	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0[41]	http://datos.santander.es/dataset/?id=paradas-bus
Taxi stops	Location of cab stand signs located within the municipal territory of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=paradas-taxis
Parking spaces for people with reduced mobility	Information related to the parking spaces enabled for people with reduced mobility or handicapped, and their geographical position.	RDF HTML JSON N3 XML	CC BY 4.0	http://datos.santander.es/dataset/?id=plazas-pmr

[41] Attribution 4.0 International; a Creative Common license. https://creativecommons.org/licenses/by/4.0/deed.en

		TURTLE CSV ATOM JSONLD		
Induction loop data	Measurements made by the magnetic loops used by the Municipal Traffic Control Center to regulate traffic and traffic light programming. Real Time update provided each minute.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=datos-trafico
Location of the magnetic loops	Location of the magnetic loops used by the Control Center Municipal Traffic to regulate traffic and traffic lights programming.	SHP	CC BY 4.0	http://datos.santander.es/dataset/?id=datos-trafico
Induction loop HISTORICAL data	Historical data from the last seven days of measurements carried out by the magnetic loops.	CSV	CC BY 4.0	http://datos.santander.es/dataset/?id=historico-semanal-de-mediciones-de-trafico
Santander Cultural Agenda	Information on the Cultural events programmed within the Municipality of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=agenda-cultural

Points of Interest Santander City	Points of Interest Santander City (e.g. museums, theatres, monuments, beaches, parks, etc.).	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=puntos-de-interes
Meteorological data	Real time measurements from different sensors located in the city of Santander related to the environment, light, noise, temperature, etc.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=sensores-ambientales
Movil sensors_environmental measurements	Real time information of the environmental measurements made by the sensors equipped in the Public Transport Vehicles, and maintenance of the city used by the Santander City Council for its daily management. The data is provided by Orion Context Broker, from the Fiware platform.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/resource/?ds=sensores-moviles

Calendar of Santander Public Holidays	Calendar of Holidays defined in the Municipality of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=calendario-laboral
Shops dedicated to retail sales	Information about shops located in the Municipality of Santander mainly dedicated to retail sales.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=comercios
Cinemas in the city of Santander	Catalogue of existing cinemas in the city of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=cines

Museums in the city of Santander	The list of museums available in the city of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=museos
Cultural galleries in the city of Santander	The list of the cultural galleries available in the city of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=galerias-culturales
Santander Gardens	Geographical representation of the Gardens located within the Municipality of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=jardines

Santander Parks	Geographic representation of the parks located within the municipality of Santander.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=parques
Municipal Buildings	Inventory of Municipal Buildings in geographic format.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD WKT	CC BY 4.0	http://datos.santander.es/dataset/?id=edificios
Building Permit applications	Building Permit applications made through the General Registry of the Santander City Council.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=solicitudes-licencias-obra

Road network of the Municipality of Santander	Geographical data relating to the road network of the Municipality of Santander	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD SHP	CC BY 4.0	http://datos.santander.es/dataset/?id=callejero
Waste Containers	Information about the waste containers operating in the Municipality of Santander such as position, capacity, and even measurements of the built-in sensor.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=residuos
Bike lane network	Bike lane sections spread over municipality of Santander	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=carril-bici

Bike sharing facilities occupancy	Real Time update Stations existing municipal bicycle rental. The status of rental bike stations, number of existing and free anchors.	RDF HTML JSON N3 XML TURTLE CSV ATOM JSONLD	CC BY 4.0	http://datos.santander.es/dataset/?id=estaciones-bicicletas

Flanders pilot site

Title	Description	Format	License	Link
GRB 2D Base layer	Geographical and characteristic information of buildings, plots, roads and their layout, waterways, railways and the road network.	Shape	-	https://www.geopu nt.be/catalogus/dat asetfolder/7c82305 5-7bbf-4d62-b55ef85c30d53162
Digital height model Flanders (up to 1 meter)	Digital Terrain Model (DTM) of the ground level in grid format with a ground resolution of 1 meter.	WMS	-	http://www.geopu nt.be/catalogus/dat asetfolder/f52b1a1 3-86bc-4b64-8256-88cc0d1a8735
Orthofoto high scale (10cm)	Large-scale orthophoto mosaic (ground resolution: 10 cm) of the digital aerial images obtained simultaneously with the LiDAR height data.	WMS		https://www.geopu nt.be/catalogus/dat asetfolder/dbbeddd 4-0452-4413-9f2afa47a4f98e55
MAGDA Services	National register, information about age and family composition	-	-	https://www.ibz.rrn .fgov.be/nl/rijksregi ster/
Loop-based traffic data (1-minute delay)	This data is provided by the inductive detection loops, mainly on highways in Flanders. The data controllers are Agentschap Wegen en Verkeer (Roads and Traffic Agency) and Vlaams Verkeerscentrum (Flemish Traffic Centre). The traffic data contained are the number of vehicles and the average speeds, divided in 5	XML	CC BY 4.0	https://opendata.vlaanderen.be/dataset/meten-in-vlaanderen-minuutwaarden-verkeersmetingen

	vehicle classes, aggregated per minute and the location of the measurement points.			
Air quality - HQ sensors (multiple elements)	Information about ambient air quality	CSV	CC BY 4.0	https://www.irceline.be/en/documentation/open-data
Luftdaten[42] - air quality sensors	Environmental data collected by sensors (sensor location, PM10, PM2.5, humidity, temperature, noise).	CSV	Open Database	https://archive.sensor.community/

[42] Luftdaten (Now Sensor.Community) is a citizen science initiative to measure air quality. the initiative is mainly active in Europe and generates a continuously updated air quality map from the data transmitted by the contributors. - https://sensor.community

Helsinki pilot site

Title	Description	Format	License	Link
3D model of Helsinki	A semantic city information model and a visually high-quality reality mesh model	GML	CC BY 4.0	https://hri.fi/data/en_GB/dataset/helsingin-3d-kaupunkimalli/resource/577f4286-7162-42e9-8ffe-52632228569e?inner_span=True
Helsinki Energy and Climate Atlas	Heating, electricity and water consumption data for 2015, 2016, 2017 AND 2018	XLSX	CC BY 4.0	https://hri.fi/data/en_GB/dataset/helsingin-3d-kaupunkimalli/resource/48afc399-9d3c-4976-8375-daf5863ad7af
Real-time swimming water temperature in Helsinki	swimming water temperature detected using sensors, in Helsinki.	JSON CSV	CC BY 4.0	https://hri.fi/data/en_GB/dataset/veden-reaaliaikainen-lampotila-helsingin-uimarannoilla
Up-to-date LAM[43] measurement data	Measurement data for LAM stations. Amount of traffic in each direction for each LAM station, and measured average speed in both directions. The information is updated almost in real time, but the outgoing message is cached for one minute. More info at https://www.digitraffic.fi/tieliikenne/	JSON	CC BY 4.0	https://tie.digitraffic.fi/api/v1/data/tms-data
Park & Ride parking	Information about parking spaces in Helsinki	JSON GeoJSON		https://p.hsl.fi/api/v1/facilities.json

[43] LAM: Liikenteen Automaattinen Mittaus (Automatic Traffic Measurement)

Public transport stops	Location of public transport stops	GraphQL JSON	CC BY 4.0	https://digitransit.fi/en/developers/apis/1-routing-api/stops/
Public transport statistics	Helsinki region transport passengers by station	CSV KML ESRI Shapefile GeoJSON	CC BY 4.0	https://hri.fi/data/en_GB/dataset/hsl-n-nousijamaarat-pysakeittain
City bikes	Location of bike rental stations and bike availability	XML GraphQL JSON	CC BY 4.0	https://api.digitransit.fi/routing/v1/routers/hsl/bike_rental
Cyclist information	The number of cyclists in Helsinki, starting from the beginning of 2014. This dataset is updated every 6 months.	CSV	CC BY 4.0	https://hri.fi/data/en_GB/dataset/helsingin-pyorailijamaarat
Street network traffic stats	The data set contains data on the volume, speed and vehicle distribution of motor vehicle traffic in Helsinki.	CVS	CC BY 4.0	https://hri.fi/data/dataset/liikennemaarat-helsingissa
Street network accident stats	Locations, severity and types of accidents in Helsinki since 2000.	CVS	CC BY 4.0	https://hri.fi/data/fi/dataset/liikenneonnettomuudet-helsingissa
Linked Events	Categorized data on events and places from the City of Helsinki, including Helsinki Marketing, Helsinki Cultural Centres and the Helmet metropolitan area public libraries.	JSON-LD	CC BY 4.0	https://api.hel.fi/linkedevents/v1/
Public spaces	Access to public spaces and other public resources.	JSON	-	https://api.hel.fi/respa/v1/
Library card users	Statistics of the Helsinki City Library for different years on Library Card users by sex	Excel file	CC BY 4.0	https://hri.fi/data/fi/dataset/helsingin-kaupunginkirjastossa-kirjastokorttiaan-k-ytt-neet

	and age group. The time series starts from 2007.			
City map services	Orhomosaic, aerial photographs, city models, POI layer	GML KML Excel file AutoCAD DXF GeoPackage MapInfo MIF MapInfo TAB Shape	CC BY 4.0	https://kartta.hel.fi/?setlanguage=en#
Helsinki metropolitan area service map	Information on the service points and services offered by the cities of Helsinki.	CSV JSON	CC BY 4.0	https://hri.fi/data/en_GB/dataset/paakaupunkiseudun-palvelukartan-rest-rajapinta

8 Annex 2 – Main components of URBANAGE Platform interacting with the Data Management Layer

This section briefly describes the main logical components of the URBANAGE Platform that interacts with the Data management Layer.

- **URBANAGE UI**: this component represents the front-end of the entire URGANAGE Platform and integrates the UIs provided by the components of the platform itself. Its aim is to offer a unified and uniform view to the users. It also provides UIs for the management of various aspects of the system like user management, user account settings etc. User interfaces of the components of the Data Management Layer are exposed through the URGANABE UI (e.g. administration UI of the Data Gateway)
- **City Information Model**: this component offers management capabilities for urban models following the CityGML [19] standard that contains semantically rich, hierarchically structured, multi-scale urban objects facilitating complex GIS modelling and analysis tasks, far beyond visualization. A connection between the City Information Model and the Context Information (managed by the Context broker) is foreseen.
- **Workflow Management**: This component allows to build, execute, and monitor workflows for the orchestration of internal processes of the URBANAGE Platform. It allows to define and schedule workflows and to monitor their execution and offers a wide range of integration options to interact with different protocols/systems. It also provides a graphical user interface (GUI) that will be is part of the URBANAGE UI. The collection of static information coming from data repositories leverages the Workflow Management to schedule and automate collection processes.
- **Message Bus**: This component offers a messaging system that assists the synchronous and asynchronous communication within the URBANAGE Platform. The Context Broker notifies real-time updates related to context information to the Message Bus (through the Context Information Data Bridge component), so to communicate to the AI Algorithms and simulation tools and to the Big Data Analytics component the needed information.
- **AI Algorithms and simulation tools**: these components offer simulations and route planning capabilities making use of optimization modules based on multi-objective algorithms and ML/DL algorithms to achieve different descriptive, predictive and prescriptive functionalities. In addition to the information transmitted through the Message Bus, these components can access static information (made available through Data Repositories Federator) and the context information (made available through the Context Broker)
- **Big Data Analytics components**: these components include the ones devoted to the storage of big data (i.e. the URBANAGE Data Lake) and to their analysis. In detail, the first provides a storage facility

for the data collected and enables further analysis by the Big Data Analytics and AI components. The second offers the capabilities to analyse the data collected and stored, by leveraging various techniques like machine learning and data mining to deliver descriptive, predictive and prescriptive analytics. As for the AI algorithms and simulation tools, in addition to the information transmitted through the Message Bus, these components can access the static information (made available through Data Repositories Federator) and the context information (made available through the Context Broker).

9 Annex 3 – Data Management Layer baseline tools

This section provides more technical details about the baseline tools used and/or customized to build together the Data Management Layer.

Apache Airflow

Airflow is an *open-source* tool to programmatically author, schedule, and monitor workflows. It is one of the most robust platforms used by data engineers for orchestrating workflows or pipelines. This tool allows to easily visualize data pipelines' dependencies, progress, logs, code, trigger tasks, and success status. Airflow provides an administrative user interface for easy visualization of running pipelines, monitoring their progress, and troubleshooting issues when needed. In Airflow, pipelines are configured as Python code. An Airflow pipeline is a Directed Acyclic Graphs (*DAGs*)[44] of tasks. A task represents a node of a defined DAG that can use diverse operators to interact with a wide range of protocols/systems and integration options provided by Airflow. It can be connected with multiple data sources and can send messages/alerts via different protocols when a task has been completed or a task execution has been processed with errors. Apache Airflow is distributed, scalable and flexible, making it well suited to handle the orchestration of complex business logic. Its modular architecture is composed of diverse component' types that interact with each other to orchestrate data pipelines as reported in Table 5.

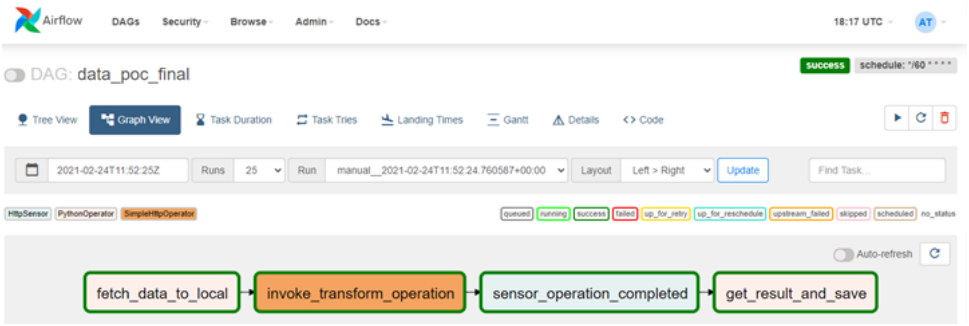
Table 5: Apache Airflow Architecture Components

Component Name	Component Description
Web Server	This is the administrative User Interface of Airflow, that can be used to get an overview of the overall health of different Directed Acyclic Graphs (DAG) and also help in visualizing different components and statuses of each DAG. The web server provides also capabilities to manage users, roles, and different configurations for the Airflow setup. The web server provides also a sets of *REST APIs* that can be used to perform various tasks like triggering DAGs or getting the status for each task instance,
Scheduler	This is the most important part of Airflow, that orchestrates various DAGs and their tasks, taking care of their interdependencies, limiting the number of runs of each DAG so that one DAG doesn't overwhelm the entire system, and making it easy for users to schedule and run DAGs on Airflow.
Executor	Executors are the components that actually execute tasks. There are different types of executors that come with Airflow, such as *Sequential Executor, Local Executor, Celery Executor* and the *Kubernetes Executor*. An executor can be selected based on a specific use case needed. For example, a Celery Executor, based on python celery, is widely used to process asynchronous tasks while Sequential Executor is used only to run one task instance at a time (See Apache Airflow documentation for more details).

[44] A Directed Acyclic Graph (DAG) is the core concept of Airflow; it allows to collect Tasks together, organise their dependencies and relationships [20]

Metadata Database	Metadata Database stores metadata about DAGs, their runs and other configurations like users, roles and connections. This information, such as DAGs' statuses and their runs, is visualized on web server UI and it is updated by the scheduler. Airflow supports a variety of database management systems (DBMS) for its metadata store.

Figure 19: Airflow UI showing a DAG graphical representation



Presto DB

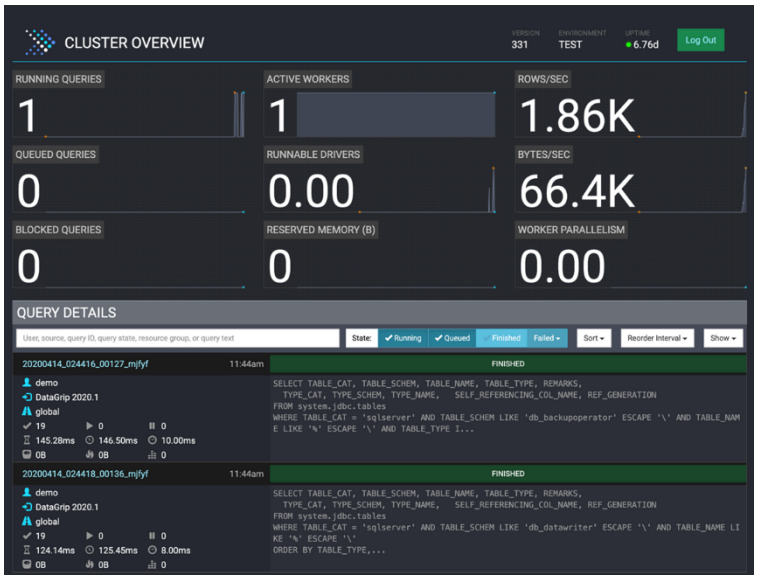
Presto DB is an open-source software released under the Apache License that works as high performance, distributed SQL query engine for big data. Presto has been designed to handle data warehousing and analytics: data analysis, aggregating large amounts of data and producing reports. Its main peculiarity consists in the capability of combining data from multiple sources with a single query. Its architecture allows users to query a variety of data sources and it offers connectors to data sources including files in *Hadoop Distributed File System, Amazon S3, MySQL, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Apache Kudu, Apache Phoenix, Apache Kafka, Apache Cassandra, MongoDB and Redis*. Concerning architecture aspects there are two diverse types of servers in Presto: Coordinators and Workers (Table 6).

Table 6: Presto server types

Server Type	Description
Coordinator	The server responsible for parsing statements, planning queries and managing Presto worker nodes. It is a mandatory server in Presto architecture that must have a coordinator. Coordinator communicates with workers and clients using REST APIs.
Worker	The server responsible for executing tasks and processing data. Workers communicate with other workers and the coordinator using REST APIs.

Presto provides a web user interface for monitoring and managing queries (Figure 20). The UI offers a list of queries along with information like unique query ID, query text, query state, percentage completed, username and source from which this query originated.

Figure 20: Presto user interface



The currently running queries are at the top of the page, followed by the most recently completed or failed queries. Table 7 summarises the possible query states.

Table 7: Presto DB possible query states

Status	Description
QUEUED	Query has been accepted and is awaiting execution
PLANNING	Query is being planned.
STARTING	Query execution is being started.
RUNNING	Query has at least one running task.
BLOCKED	Query is blocked and is waiting for resources (buffer space, memory, splits, etc.).
FINISHING	Query is finishing (e.g. commit for auto commit queries).
FINISHED	Query has finished executing and all output has been consumed.
FAILED	Query execution failed.

From a technical point of view, Presto requires a Java Virtual Machine (JVM) to run, whereas its UI is based on the React framework[45]. The Presto source code is available on the official GitHub repository[46].

[45] https://reactjs.org/
[46] Presto DB GIT repository - https://github.com/prestodb/presto

Idra

Idra is an Open Data Federation Platform licensed under Affero General Public License (AGPL version 3) that provides functionalities to search, discover and visualize datasets from heterogenous Open Data Management Systems (ODMS), working as unique point of access for resources coming from federated ODMS. This platform has been developed as a Java Enterprise Edition (J2EE) application and it can be installed in two diverse ways as a *WAR package* deployed in an application server or as a *Docker containerized* environment. In the *installation* section of the official documentation [20] are reported further information about the two installation processes. Idra uniforms representation of collected open datasets, thanks to the adoption of international standards (DCAT-AP) and provides a set of RESTful APIs to be used by third party applications. Indeed, Idra provides a set of Restful APIs to interact with the tool and its provided functionalities. These APIs are developed following the *OpenAPI* specification [21]. These APIs are grouped into these different groups reported in Table 8.

Table 8: Idra APIs groups

APIs Group Name	APIs Group Description
Administration APIs	APIs devoted to features permitted only to the administration users.
End-user APIs	APIs devoted to provides platform features to the end users.
Federation APIs	APIs devoted to the management of federated catalogues.

Further information about Idra APIs is reported in the official documentation. The Idra platform is responsible for collecting metadata of Open Data from federated ODMS catalogues and then for translating them into a common and uniform format. Moreover, it manages Linked Open Data (LOD), importing them into a specific repository to perform queries. About federation it supports different technologies such as DKAN, CKAN, ORION, SOCRATA etc. From the technical point of view, Idra is composed of two components: a backend module and a web portal. The web portal provides a user-friendly interface for interaction between end users and platform. The backend module exposes a set of RESTFul APIs to interact with the provided diverse functionalities invoked by end-users and/or third party.

Beyond that, Idra provides functionalities to perform federated *metadata search* among federated catalogues and provides capability to *filter results* of metadata search by using Tags, Data Formats, Licenses and other fields as depicted in the two figures below. Another relevant feature is related to data visualization. End users can create a graphical representation of a selected resource by using a Datalet [22], a tool integrated with Idra that allows to create rich and reusable visualization of open data.

Figure 21: Idra user interface

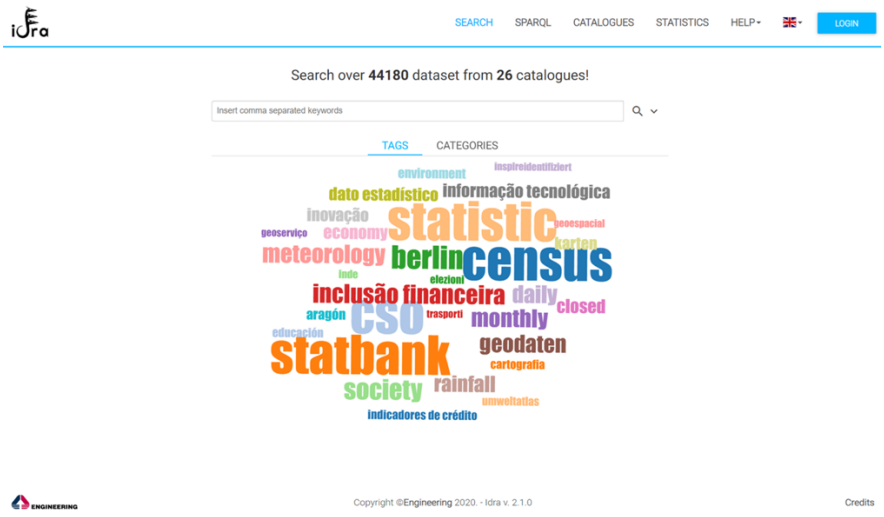
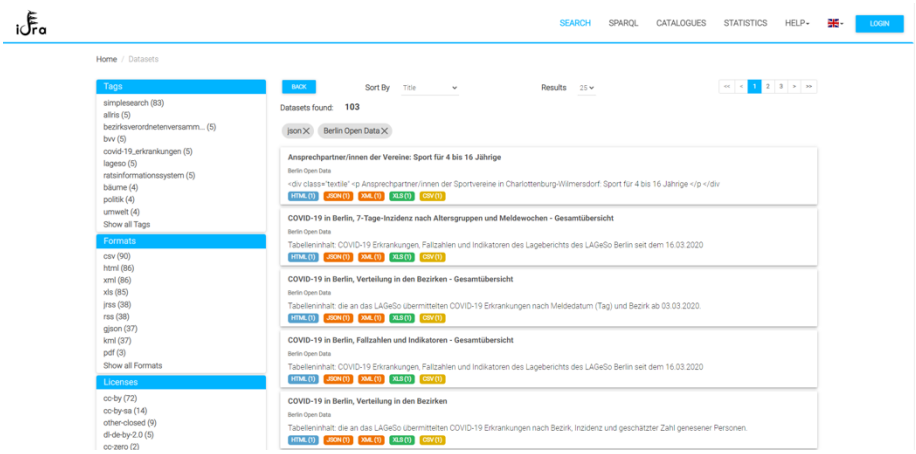


Figure 22: Filtered results from a federated metadata search on Idra



Data Model Mapper

The Datamodel Mapper[47] is licensed under Affero General Public License (AGPL) version 3 and has been initially developed in the context of the SynchroniCity project[48]. This tool, which requires NodeJS[49], is able to translate structured input data into NGSI compliant entities. Data Model Mapper is able to convert several file formats such as CSV, JSON and GeoJSON according to different data models. For this purpose, the tool uses a mapping file, which is a well-formed JSON file, to know which rules it must adopt for mapping each source field of the parsed row/object in the destination fields. During the conversion process, the tool performs the steps reported in Table 9.

[47] Data Model Mapper - https://gitlab.com/synchronicity-iot/data-model-mapper
[48] Synchronicity Project - https://synchronicity-iot.eu/
[49] https://nodejs.org

Table 9: Data Model Mapper steps

Step Name	Step Description
Parsing	Parsing of input file by converting it into a row/object stream.
Streaming	Each row or object that comes from the stream is converted to an intermediate object.
Mapping	Conversion of the intermediate object to an NGSI entity, according to a specific target Data Model by using the input JSON Map.
Validation (and report)	Validation of resulting object against the JSON schema corresponding to the target Data Model (and production of a report file with validated and unvalidated objects).
Writing	Delivery of validated objects to the configured Orion Context Broker writer and/or to a local file writer.

The input file to be converted can contain either rows, JSON objects or GeoJson features, each of them representing an object to be mapped to an NGSI entity, according to the selected data model passed to the tool as an input parameter. For correct mapping, the tool needs the mapping instruction to know how to map each source field into the destination fields. The mapping instructions consist of a JSON file, which is a collection Key/Value pairs. More detail about mapping has been reported on the official documentation [23]. Once Data Model Mapper has been successfully configured by editing paths and variables that are in the global configuration file (*config.js*), the tool is ready to be used. Figure 23 depicts an example of a mapper command with the CLI arguments needed to perform a specific conversion.

Figure 23: Example mapper command on Datamodel Mapper

Example:

```
node mapper -s "path/to/sourcefile.csv" -m "path/to/mapFile.json -d "WeatherObserved"
```

Table 4: Data Model Mapper CLI arguments

Short Name	Long Name	Description
-s	--sourceDataPath	The path of the source file
-m	--mapPath	The path of JSON Map with extension ".json"
-d	--targetDataModel	The name of target **Data Model** that have to match with those contained in the "/dataModels" folder

Orion Context Broker

Orion Context Broker[50] offers functionalities to manage context information and to execute different operations like *subscriptions, notifications and data gathering*. It manages the entire life cycle of context information dispatching real-time information received from IoT Agents following an approach based on publish-subscribe pattern. The Context Broker receives data from the IoT Agents and from the IT connectors and dispatches all received information, which is represented under the form of NGSI-LD entities, to the subscribed entities. To execute its job the context broker needs to interact with IoT Agents, which are FIWARE Generic Enablers that facilitate the connection of IoT devices, and more specifically it needs to gather context information from them. Moreover, these IoT agents can also trigger actuations in response to context updates in a flexible way by supporting different protocols such as LoRaWAN, Sigfox, Lightweight M2M and UltraLight. Orion Context Broker is developed in C++ language for performance reasons, and it offers an implementation of both NGSI-LD [17] and NGSIv2 [24] APIs.

Cygnus

Cygnus is licensed under Affero General Public License (AGPL) version 3 and persists NGSI-LD entities managed by the Context Broker creating an historical view of context information and an historical view of their related attributes. Cygnus plays a role of a connector between the NGSI source of data (i.e., Orion Context Broker) and many FIWARE storages and tools such as Cosmos Big Data (i.e., Hadoop), STH Comet and other non FIWARE storages and tools such as MySQL, Kafka, Carto, etc.

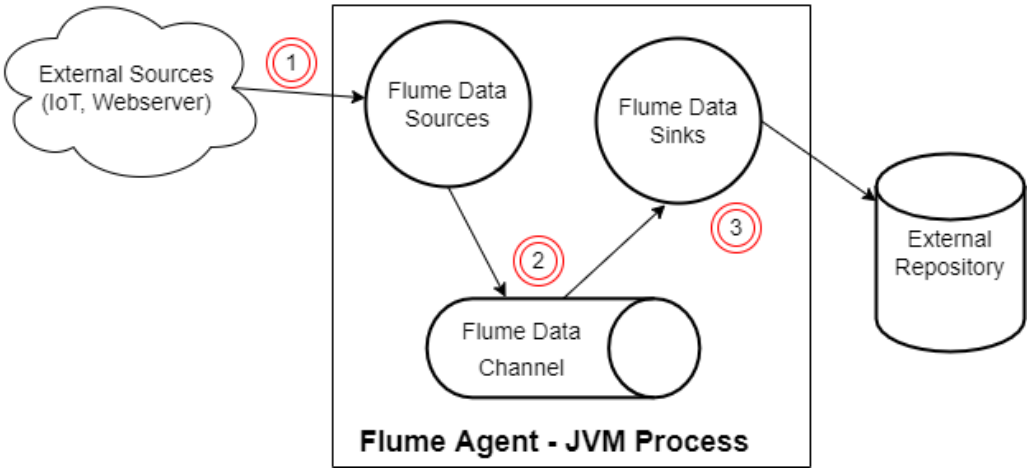
Indeed, this component is a connector in charge of persisting certain sources of data in certain configured third-party storages in diverse technologies, creating a historical view of these data. From the technical point of view, Cygnus is based on Apache Flume[51]. Apache Flume has a simple and flexible architecture based on streaming of data flows. It works as a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of data with reliability and fault tolerant mechanisms. Cygnus is designed to run a specific Flume agent per source of data. A Flume agent, which is a key concept of Cygnus, is a JVM process with three main components needed to propagate events started from external sources that are Flume Source, Flume Channel and Flume Sink.

Figure 24 depicts the Flume architecture with these main components and the following steps:

1. Events generated by external source are consumed by Flume Data Source. (These events are sent in a format recognized by target source.)
2. Flume Source receives an event and stores it into one or more channels. (A channel may use a local file system to store these events.)
3. Flume sink removes the event from a channel and stores it into an external repository.

[50] FIWARE-ORION - https://fiware-orion.readthedocs.io/en/master/
[51] Apache Flume - https://flume.apache.org/

Figure 24: Flume Agent components



Following the above steps, it can be summarized that a Cygnus agent is composed of a listener or source in charge of receiving data, a channel where the source puts these data and finally a sink which takes Flume events from the channel to persist them into a third-party storage. Further details about API methods for Cygnus and integration examples are available in the official documentation [18].

IoT Agents

IoT Agents support different protocols to gather context information provided by IoT devices through IoT connectors. A generic IoT Agent translates IoT-specific protocols into the NGSI context information protocol, that is the FIWARE standard data exchange model. In particular, an IoT Agent is a component that lets a group of devices send their data to and be managed from a Context Broker using their own native protocols. They should also be able to provide common services to the device programmer and to deal with security aspects of the FIWARE platform such as authentication and authorization of the channel. There are many existing IoT Agents for diverse communication protocols and data models. Table 10 summarises a list of IoT Agents and the supported communication protocols with a brief description. For further information about IoT Agents refer to the official documentation [25].

Table 10: IoT Agents and communication protocols

IoT Agent	Protocol	Description
IoTAgent-JSON [26]	MQTT	Abridge between HTTP/MQTT messaging with a JSON payload and NGSI
IoTAgent-LWM2M [27]	Lightweight M2M	A bridge between the Lightweight M2M protocol and NGSI
IoTAgent-UL [28]	UltraLight 2.0	A bridge between HTTP/MQTT messaging, with an UltraLight2.0 payload, and NGSI
IoTagent-LoRaWAN [29]	LoRaWAN	A bridge between the LoRaWAN protocol and NGSI

TRUE Connector

The aim of this component is to facilitate secure and standardized data exchange and data linkage in a trusted ecosystem. Indeed, the FIWARE TRUE Connector enables the trusted data exchange to be active part of an IDS Ecosystem[52] , i.e. a virtual data space leveraging existing standards and technologies.

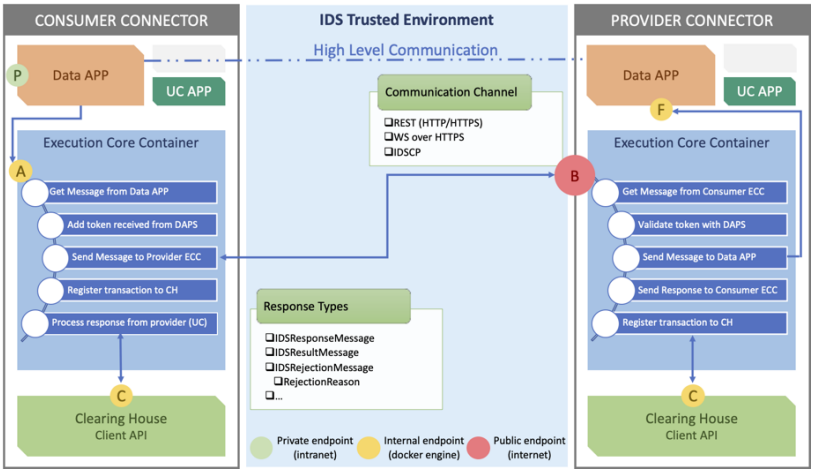
The FIWARE TRUE (**TRU**sted **E**ngineering) **C**onnector [30] for the IDS (International Data Space) ecosystem, released under AGPL version 3 license, can be easily customized to fit a wide spread of scenarios. This customization can be done thanks to the internal separation of *Execution Core Container* and *Data App*. The connector is easily integrable with a lot of existing IDS services and it is extremely configurable in terms of internal/external data format such as multipart/mixed, multipart/form, http-header. Moreover, it can be configured to supports/uses a huge number of protocols such as HTTP, HTTPS, Web Socket over HTTPS, IDSCPv2. The TRUE Connector is composed of three main components reported in Table 11.

Table 11: TRUE Connector components

Component	Description
Execution Core Container **(ECC)**	Representing the connector exchanging data.
FIWARE Data Application	It is in charge of processing incoming request and provided the relative responses.
Usage-Control **(UC)** Data Application	It will check if who are requesting the data has the grants to use that in a well-defined policy.

The TRUE connector can be run as consumer (send the request to the provider to obtain some data) or provider (provide the data to the consumers if allowed from the policies in UC). Figure 25 depicts the components of the TRUE Connector and the possible roles (consumer and provider).

Figure 25: TRUE Connector components[53]



[52] IDS Ecosystem - https://opcfoundation.org/markets-collaboration/ids/

[53] Image from FIWARE TRUE Connector repository - https://github.com/Engineering-Research-and-Development/fiware-true-connector/

Amnesia

Amnesia is a tool developed by OpenAIRE for Data Anonymization that allows end users to anonymize sensitive data in order to share them with a broad audience. Amnesia offers to the users an assisted anonymization process by visualizing the candidate solutions and allowing to choose and customize the most convenient one. For this purpose, it provides a user-friendly interface to get anonymous data just following the below (macro) steps:

1. **Data Import**: Import the original data (e.g., a simple text file with any type of delimiter).
2. **Creation of the generalization hierarchies**: The user defines the rules for anonymising the data.
3. **Selection/Preview of data to be anonymized and anonymization**: The user chooses the algorithm and the parameters to perform anonymization.
4. **Download of the anonymized data**: The anonymized data can be now downloaded.

From a technical point of view, Amnesia requires a Java Runtime Environment to run, and its backend is implemented by using Spring framework. Its components offer a REST API that handles anonymization requests issued by the web interface. Amnesia uses a temporary local storage for the anonymization purposes and final results are returned via the REST interface. Further information about REST APIs is available on Amnesia Git repository [31].