Co-financed by the European Union
Connecting Europe Facility

Water Health Open knoWledge (WHOW)

Design of the technical services for knowledge graph management

Project number	**Agreement number: INEA/CEF/ICT/A2019/2063229 Action No: 2019-EU-IA-0089**
Project acronym	**WHOW**
Project title	**Water Health Open knoWledge**
Project duration	**36 months (01/09/2020 – 31/08/2023)**
Programme	**Connecting Europe Facility (CEF) Telecom**

Activity title	**Technical Architecture Implementation**
Deliverable number	**[4.#7]**
Version (date)	**0.1 (07/10/2021)**
Due date	**2021/11/30**

Responsible organisation	**Institute of Cognitive Sciences and Technologies of the Italian National Research Council - ISTC-CNR**
Editors	**Anna Sofia Lippolis, Giorgia Lodi, Andrea G. Nuzzolese**

Abstract	**This deliverable contains the description of the design of WHOW technical architecture to be deployed at each data provider.**
Keywords	**Linked Open Data, RDF, knowledge graph, data consumption, data provision, technical use cases, technical design**

Editors

Anna Sofia Lippolis, Giorgia Lodi, Andrea Giovanni Nuzzolese

(Institute of Cognitive Sciences and Technologies of the Italian National Research Council - ISTC-CNR)

Reviewers

Luigi Asprino (University of Bologna), Alessandro Russo (BUP srl - CNR Spinoff)

Contributors

From the project - Giovanni Peditto (ARIA SpA), Elio Giulianelli (ISPRA)

From co-creation programme - Angelo Gulina (GitLab), Giovanni Pirrotta (Open Data Sicilia community) Matteo Busanelli (Imola Informatica), Marco Panebianco (ARIA SpA), Azzurra Pantella (Umbria Digitale), Giulio Settanta (ISPRA), Andrea Borruso (OnData association), Stefano Carbone (freelance consultant in the health sector), Marco Casaburi (Planetek Italia srl), Paolo Francescangeli (Italian National Institute of Statistics - ISTAT)

Acknowledgement

This work was partially supported by the European Commission (EC) through the Connecting Europe Facility (CEF) programme under the WHOW project WHOW (grant agreement no. INEA/CEF/ICT/A2019/2063229).

Disclaimer

The sole responsibility of this publication lies with the author. The European Union is not responsible for any use that may be made of the information contained therein.

Confidentiality

Change Log

Version	Date	Organisation	Description
0.1	2021/06/08	ISTC-CNR	Creation of the Table of Content
0.2	2021/09/05	ISTC-CNR	Included first content on use cases (Section 3)
0.3	2021/09/30	ISTC-CNR	Completion of the first version of Section 3, Executive Summary, Introduction and Data consumers and producers section
0.4	2021/10/15	ISTC-CNR	Preliminary content for section 4
0.5	2021/10/29	ISTC-CNR	Implementation of some comments of WHOW partners, completion of section 4 and conclusions.
0.6	2021/11/02	ISTC-CNR	Document fully completed and ready for internal and external revision
0.7	2021/11/17	ISTC-CNR	First revision of the document in order to respond to comments of external reviewers and co-creators
0.8	2021/11/24	ISTC-CNR	Revision of some part of the document based on the received reviews

Executive summary

The main goal of the WHOW project is to build an open and distributed knowledge graph that is capable of integrating and standardising heterogeneous data of the environmental and health domains coming from several data sources and available in different formats and structures.

In particular, through identified business use cases, the project aims at creating a large knowledge base capable of linking data about water consumption and pollution with health parameters (e.g., disease spreading). The ultimate goal is fostering the creation of innovative applications, services and studies on top of the WHOW knowledge graph.

Besides the business use cases that are going to be extensively described in deliverable D2.1 to be released at the end of December 2021, a key aspect of the WHOW project is the design of a fully distributed technical architecture for effective creation and publication of the WHOW open and distributed knowledge graph.

The technical architecture, that can be adopted by newcomers who want to contribute to the WHOW knowledge graph, consists of two main macro-elements:

- a set of semantic resources including ontologies and linked open data that are designed and produced to provide a shared semantics and standard for representing heterogeneous data of different actors and domains (i.e., water, health);
- a set of software components that, using the earlier cited semantic resources, are able to provide:
    - data consumers with tools for consuming data, and related data models, via human and machine based interaction services;
    - data providers with a technical architecture that offers software services for a sustainable data management process.

This deliverable, which represents an important milestone (#7) of the project, focuses on the design of the technical architecture and describes:

- a set of technical use cases that define how different types of users (data consumers and data providers) can interact with, and leverage the functionalities offered by, the WHOW technical architecture;
- the functional and non-functional requirements that are derived from each technical use case and used in synergies with the business use cases of the project;
- the high level view of the architecture, made up of software services and semantic resources;
- a component based design of the architecture that illustrates the interfaces used in the interactions among the architectural components;
- the process that is enabled in the construction of the ontologies and controlled vocabularies of the WHOW knowledge graph.

The technical formalism that has been used to visually represent the design of the architecture is the Unified Modeling Language (UML), a well-known and widely used modelling language in software engineering projects. Therefore, UML use case, component and activity diagrams are introduced throughout the deliverable.

TABLE OF CONTENT

List of Figures

List of Tables

1. Introduction

This document is Deliverable D4.1 - “Design of the technical services for knowledge graph management” and describes the result of two main tasks of Activity 2 and 4, respectively, foreseen in WHOW: task 2.1 - Identification of functional and non-functional requirements of the technical knowledge graph and task 4.1 - design technical services for knowledge graph management.

1.1 Project Overview

The WHOW project aims to foster the creation of the first open and distributed European knowledge graph on water consumption and quality, health parameters and dissemination of diseases to be reused for advanced analysis and development of innovative services.

The project leverages the Linked Open Data paradigm. Water related datasets from Italy and other European countries and Copernicus[1] (the European Union's Earth observation programme) will be used to support the construction of WHOW’s knowledge graph, intended as a federation of knowledge graphs deployed at each data provider willing to join the WHOW community. The knowledge graph will be documented on data.europa.eu, the official portal for European data, thanks to the adoption of shared metadata models such as (Geo)DCAT-AP[2]. Selected health related datasets from Italy will be linked to specific water datasets.

WHOW targets use cases in the creation of the knowledge graph, identifying and integrating the relevant set of indicators for Sustainable Development Goals (SDGs)[3], deploying a co-creation programme where interested stakeholders and users are engaged from the initial phases of the project.

The initiative supports the Public Open Data Digital Service Infrastructure by helping to boost the development of information products and services based on the re-use and combination of environmental data and health data on disease dissemination.

[1] https://www.copernicus.eu/en.

[2] See https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/dcat-application-profile-data-portals-europe/release/11.

[3] https://sdgs.un.org/goals.

1.2 Objectives

This document contains the architectural design and requirements for the WHOW technical architecture. The final goal of WHOW is to develop a framework to foster the creation of a large data ecosystem on water consumption and quality, health parameters and dissemination of infectious diseases to be reused by data consumers for advanced research and development of innovative services. More specifically, the architecture is designed so as to provide end-users with a modular, distributed, highly extensible and scalable framework consisting of a knowledge graph as well as software services designed based on the FAIR (findable, accessible, interoperable, reusable) principles; as such, all WHOW software components are made available as open source. Thanks to the architecture, WHOW is capable of defining a sustainable pipeline for open knowledge graph production that guarantees authoritativeness, timeliness, semantic accuracy and consistency data quality characteristics, as well as compliance of metadata with the European DCAT-AP profile and related national and thematic extensions.

The target of this document are primarily data providers and consumers involved with the monitoring of water quality and whoever else is interested in researching and participating in the development of WHOW's knowledge graph.

Fundamentally, data consumers of WHOW have the possibility to:

- exploit a set of linked open datasets deriving from different public sources on water consumption and pollution;
- use APIs through which to consume the knowledge graph and facilitate the re-use of data;
- make SPARQL queries on the knowledge graph for specific analysis and research purposes.

The five high level requirements (HLRs) introduced in Section 3 and their associated use case requirements are meant to provide a reference guideline for the implementation of the WHOW architecture.

In summary, this deliverable addresses the specification of architectural use case requirements for the design and implementation of the WHOW architecture or toolkit, as well as the design of the main software components of it.

1.3 Definitions

In the context of this deliverable we are going to use a set of key terms that are worth being clearly defined for the sake of readability and understandability.

We define **ontology** as a set of representational primitives (classes, attributes and relationships between them) with which to model one or multiple given domains of knowledge. The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application. In the broader sense, an ontology serves as a framework of

knowledge representation about the world. Its precise, clear, formal semantics enables applications and software agents to process the information described by the ontology and use this shared information in intelligent applications.

Resource Description Framework (RDF)[4] RDF is a standard model for data interchange on the Web**.** It represents information as triples, that is, *subject-predicate-object* where the subject and predicate are always uniquely identified by a IRI - Internationalized Resource Identifier, and the object can be represented with a IRI or a literal value. Its semantic extension, RDF Schema (RDFS)[5], provides mechanisms for describing groups of related resources and the relationships between these resources.

To retrieve data from RDF/OWL ontologies the query language **SPARQL Protocol and RDF Query Language** (**SPARQL)[6]** has been developed. It is based on the expression of triple patterns for retrieving RDF data, and produces result sets, or RDf graphs, as output.

We define **Linked Data** as a set of design principles for sharing machine-readable interlinked data on the Web. It is one of the core pillars of the Semantic Web, also known as the **Web of Data**, and is based on four design principles according to Tim Berners-Lee [3]: (i) the use of Uniform Resource Identifiers (URIs) to refer to any resource, (ii) the use of HTTP URIs for effective retrieval of said resources, (iii) the use of RDF and SPARQL standards to respectively publish and retrieve data, (iv) the interconnection of URIs to interlink existing data in a shared network of machine-processable meaning.

With the term **knowledge graph** we mean a knowledge base which, thanks to the graph data structure, is able to link both descriptions of real-world entities with their relations and facts about these entities and relations. A semantic knowledge graph is usually represented in RDF, thus allowing for the sharing of a fluent representation of various types of data and content.

As for **open data**, and in particular **open government data**, we embrace the definition of the European Data Portal; therefore, Open (Government) Data refers to the information collected, produced or paid for by the public bodies (also referred to as Public Sector Information) and made freely available to anyone who can then use, modify, and share it for any purpose.

[4] https://www.w3.org/RDF/.

[5] https://www.w3.org/TR/rdf-schema/.

[6] https://www.w3.org/TR/rdf-sparql-query/ .

1.4 Relationships with other deliverables and activities

The present deliverable D4.1 represents an important milestone (#7) of the WHOW project. It covers different tasks foreseen for the various activities composing the overall WHOW project management. In particular, Section 3 - use cases and requirements is the result of the technical activities that have been already carried out in the context of Activity 2 for task 2.1. In addition, this section can also be a valuable input for the deliverable on the definition of the business use cases to be released at the end of 2021.

The remaining parts of this document describes the main results that we obtained in the context of task 4.1 of Activity 4. The following Figure 1 shows such relationships.

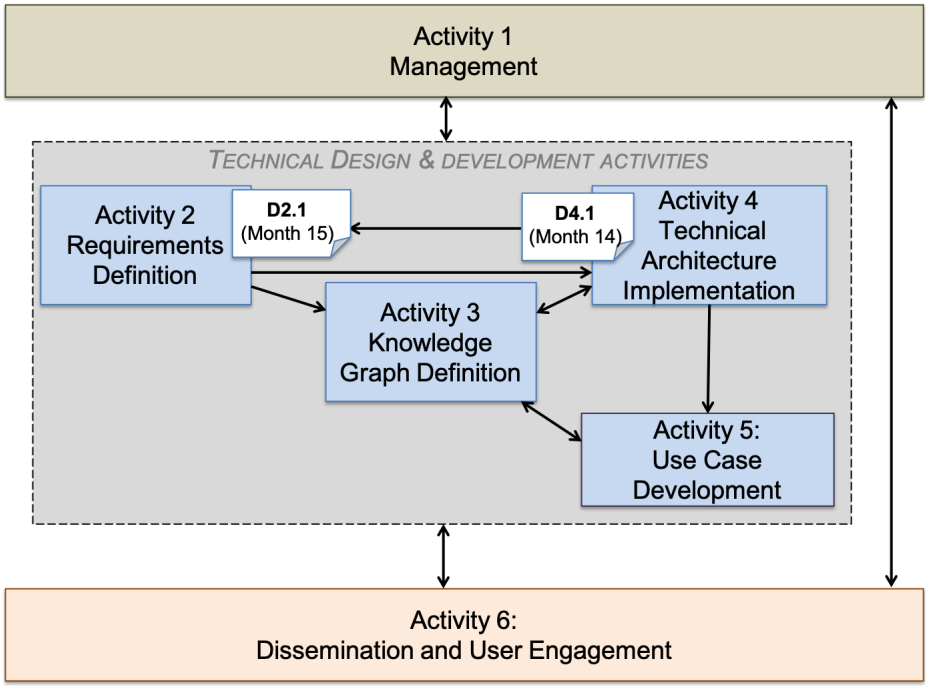

Figure 1: PERT diagram of the relationships between activities and deliverables

1.5 Structure of the document

The rest of this document is structured as follows. Section 2 provides an overview of the stakeholders of the WHOW architecture, dividing them into data providers and data consumers. The needs of these users are considered in the methodology that allows us to elicit the requirements for the design of the WHOW architecture. The methodology is described in Section 3. Section 4 describes the technical use cases. For each use case, section 4 provides scenarios, the conditions that characterise the use case and the related functional and non-functional requirements.

Section 5, based on the identified requirements introduces the Linked Open Data reference architecture of the WHOW project. A general high level view of it is presented along with a more detailed formal description of its layers and components using UML. Finally, Section 6 concludes the deliverable.

2. Stakeholders

In the context of the WHOW project, a stakeholder represents a group or organization or a single person (e.g., a civic hacker) that has interests or concerns in the WHOW data offering and technical infrastructure. For the scope of the use cases identification and requirements of this specific deliverable we divide the stakeholders into two main macro categories:

- **data providers**: they own and/or manage water and health related data. As such, they can offer for free their data via the WHOW knowledge graph, or enrich and link their data using other relevant data also available in the Web of Data.
- **data consumers**: any type of user who wants to use or explore the WHOW knowledge graph also to develop new products, services and new studies.

2.2 Data providers

The main data providers of the WHOW project are: i) ARIA S.p.A,. in-house organisation of the Italian region Lombardy, which supports Lombardy in the digital transformation processes including the management of open data, and ii) the Italian National Institute of Environmental Research (ISPRA).

In addition to the project's partners, we identified other possible data providers of the project that can be:

- Italian public institutions that already participate in the WHOW co-creation programme. At the time of the writing of this deliverable, we have identified at least two external data providers that operate in the Italian context as local public administrations: Umbria and Friuli Venezia Giulia regions.
- European institutions that already publish open data in the environmental domain, in scope with the business use cases WHOW has identified. An example is the European Environmental Agency;
- other organisations that publish open data and document it in the European data catalogue data.europa.eu.

The data they can provide can come from different systems: from already existing (open) data catalogs, internal systems exposing public APIs or from institutional web sites where, possibly open, datasets are listed.

2.3 Data consumers

The data consumers of the WHOW project are all those actors who express an interest in the specific data on water consumption and pollution and health data. There are different categories of data

consumers potentially interested in querying WHOW data and semantic assets, and that could benefit from interacting with the overall WHOW architecture.

The categories of data consumers, also showing different technical skills and thus different levels of engagement to the resources offered by WHOW, are:

- public administrations, with public employees willing to search and use specific data in order to carry out their institutional tasks, and make available transparent information about the level of water quality and spread of diseases;
- business companies willing to re-use the data and interface through it via software tools in order to construct innovative application/services;
- Non-governmental Organizations (NGOs) interested in raising the awareness in the society about specific topics that might have an impact on people's daily life;
- research institutions, with researchers and technologists interested in carrying out research studies on a potentially vast amount of interconnected data coming also from different domains;
- data journalists who are willing to exploit the data in order to construct journalism stories on specific topics.

Most of these user categories are also part of WHOW's co-creation programme.

3. Methodology for requirement elicitation

The WHOW project requirements have been collected from the use case perspective that different stakeholders can enable when interacting with the WHOW architecture. The process involved different stakeholders (described in 3.4) and two main steps, namely 1) *requirements elicitation*, which resulted in the definition of a set of High Level Requirements (HLRs), and 2) *requirements refinement*, as each requirement was described and broken down into concrete testable system requirements via scenarios and use cases. This section introduces the overall methodology we followed for requirement specification and its application.

3.1. Requirements elicitation

The task of requirements elicitation is meant to be a first step towards information discovery, forming the precursors to the actual requirements. In fact, as a first step, High Level Requirements (HLRs) have been given identifiers, names and descriptions. The goal is to address current needs for data linking in the water and health domains, as well as posing the basis for addressing future needs and opportunities when the knowledge graph is available. For this reason, requirements elicitation techniques have involved brainstorming sessions with project partners and domain experts, and a use case approach to identify via visual aid the main actors involved and the sequence interactions between them and the system.

The initial requirements are then revised in a successive step, described in 3.2, which makes use of additional input sources as well as feedback provided by stakeholders subscribed to WHOW's co-creation programme, and (research) partners of the project.

3.2 Requirements refinement

After having assigned identifiers, names and descriptions to HLRs. Then, the refinement procedure has been applied through two main iterative phases: 1) the description of example scenarios and extraction of use cases from these, and 2) for each use case, the extraction of a set of detailed testable requirements.

This process should progress until the intention of the HLR has been completely covered in the context of WHOW.

Another important aspect is to keep the relations between HLRs and use cases and to define relations between use cases. Once the use cases have been described and consolidated, it is possible to derive the concrete system requirements.

3.3 Types of WHOW system requirements

There are two main types of WHOW system requirements:

- functional requirements;
- non-functional requirements.

The requirements are expressed using the keywords *shall/shall not* or *should/should not*, expressing two levels of requirements. In particular, the *shall* requirements express functionalities that are absolute requirements of the specification, while the *should* requirements express functionalities that under certain circumstances can be ignored, after carefully assessing the consequences. This topic will be further discussed in § 5.2. When reading WHOW requirements, a developer has to consider that the *shall* requirements indicate what is needed as core system functionalities for the WHOW stack.

To highlight what to provide a taxonomy of requirements, indentation and numeration of requirements is also provided. Thus, the *shall/shall not* and *should/should not* terminology ought not to be confused with any taxonomical relation among requirements.

3.4 Main sources of information and inspiration

The project considered different sources of information and inspiration in order to be grounded in the concrete needs of both data providers and data consumers. The requirements have thus been shared not only with the project partners, but also with other experts, so as to have a more complete vision of the architecture that can actually be a model for future research. The main sources of information and inspiration are described as follows.

State of the art analysis and their application to linked open data projects

By analyzing the current trends and best practices in this field, it was possible on the one hand to identify interesting research challenges that could be investigated by the WHOW team, and on the other hand to identify existing technologies that could be exploited in the development of the WHOW architecture, fostering reuse and interoperability. Concrete design and technology choices driven by these findings tend to belong to the architecture design rather than the requirements, although this step of state-of-the-art analysis was fundamental for HLR-02. Nevertheless, in this document the survey on the state-of-the-art can be visible on the possible technologies listed within the Linked Open Data reference architecture section and in the references section of this document. Our choice of up-to-date technologies in the field of linked open data projects to adopt for the

WHOW architecture has been supported both by external experts during the peer-review phase and the co-creation meeting.

Co-creation community

The existence, in the WHOW context, of a co-creation community of providers and practitioners made it possible to receive extensive feedback both on the requirements and on the architectural design of the WHOW architecture. By openly discussing unclear passages and current challenges in the field of linked open data development, along with issues/themes raised by the community, it was possible to make significant improvements in both the architectural requirements and adopted technologies and their presentation through this document. For this reason, all of the people who helped in this sense have been listed as contributors of this deliverable.

User stories

Partly thanks to the co-creation community and the data providers involved in the WHOW project, it was also possible to derive user stories and scenarios motivating the HLRs description in this document. Questions arisen after presenting the WHOW project at the EU Open Data Days[7] virtual conference have confirmed the usefulness and relevance of the outlined scenarios.

Learning by example and experience

Learning by example and experience has also been a necessary source of information. The knowledge of linked data projects and the discussions with project partners made it possible to foster an environment based on team working. Weekly meetings with project partners and with colleagues of the same field led to the exchange of fundamental information about best practices and current trends, but most of all allowed us to create a solid basis to efficiently work together, even when most of the work was done remotely.

[7] https://app.swapcard.com/event/eu-open-data-days/planning/UGxhbm5pbmdfNzM0MTQy.

4. Use cases identification and requirements

The high level requirements (HLR) we have identified can be grouped together in the following five macro-categories:

- **HLR-01** Data provision
- **HLR-02** RESTful-based data consumption over HTTP(s)
- **HLR-03** Query-based data consumption
- **HLR-04** Download/export Knowledge Graph
- **HLR-05** Ontology development and maintenance

In the following sections we describe in detail each of them.

4.2 Data provision

HLR ID	HLR-01
Name	Data provision
Description	In order to comply with the current standards, while acquiring data from different sources, WHOW has to include services for data cleaning, transformation, and aggregation. As data providers will make datasets available in heterogeneous formats, the platform will take into account the different possibilities to support the acquisition, ingestion and transformation of non-RDF sources and will be able to involve them in triplification processes to produce RDF datasets. Transformation of the data also implies the need to have a configuration of data update policies. The platform will also be in compliance with the reference metadata definition vocabularies, with the production and management of machine-readable, standardised dataset metadata. Finally, the platform will make it possible to extend its ontology, whose development and maintenance are described in HLR-05. In this way, the data consumer will be able to effectively query data as seen in HLR-03.
Actor(s)	Data provider agent

4.2.1 Examples and Scenarios

Scenario 1
Description. ISPRA's data architect wants to increase the findability of his/her (linked) open datasets. To this end, the data provider makes available metadata of the datasets that are compliant with European standards (and National extensions of those standards), thus enabling European and National data catalogues to harvest and then document them. **Analysis**. WHOW provides all the necessary functionalities to ensure that ISPRA's datasets are compliant with the DCAT-AP model and its national extensions, along with GeoDCAT-AP in the case of geospatial data. These metadata will also be included in ISPRA's catalogue platform and other national catalogues of references (e.g. the Italian Geospatial metadata catalogue named RNDT - Repertorio Nazionale dei Dati Territoriali).

Scenario 2
Description. A data provider wants to make available datasets in WHOW, but they are in non-RDF format only. Furthermore, the transformation of the data to support the production of RDF datasets requires the definition and management of data transformation/mapping rules for the processing of datasets in heterogeneous formats. **Analysis.** The WHOW architecture takes in input datasets available in different formats through proper plugins that are able to interface the different sources of data. WHOW semantic experts define mapping rules that describe how the original input data can be transformed in a graph represented using the RDF standard, using the ontology network being developed by WHOW semantic experts in collaboration with domain experts. The mapping rules are then executed so that to produce RDF triples, then uploaded into triple stores to be queryable by anyone.

4.2.2 Use Cases

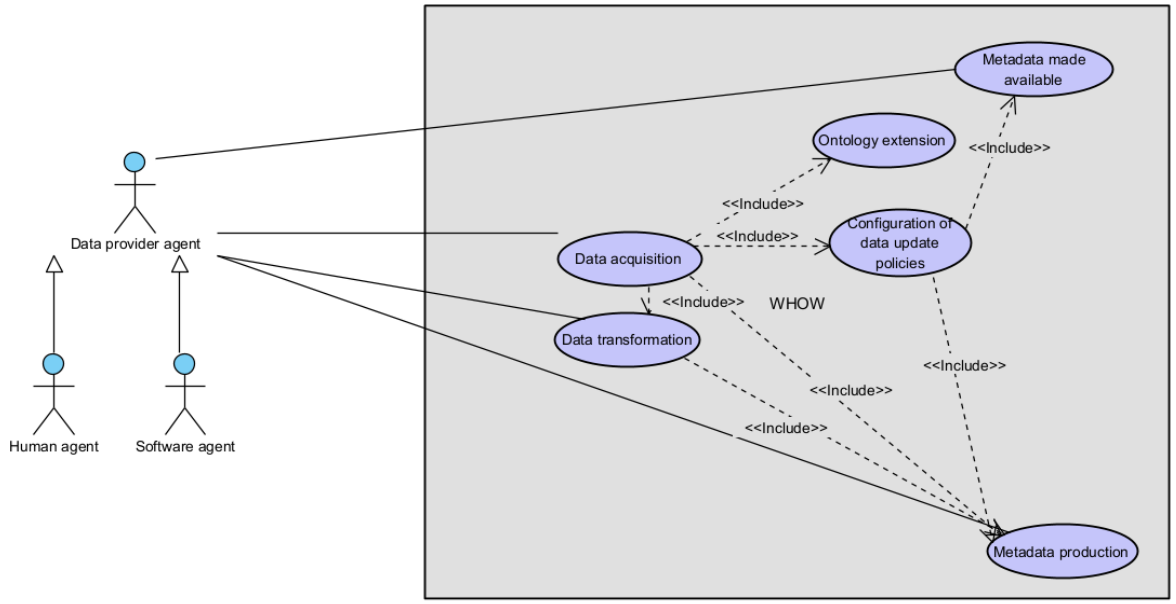

Figure 2: Use case diagram for HLR-01.

UC-0101 Data acquisition

UC-0102 Data transformation

UC-0103 Data validation

UC-0104 Configuration of data update policies

UC-0105 Metadata production

UC-0106 Metadata made available

UC-0107 Ontology extension

Use Case ID	UC-0101
Title	Data acquisition
Description	Availability of data by the data provider so that it is prepared to be cleaned, processed, and aggregated by the WHOW platform.

Actor(s)	Data provider agent (human agent or software agent)
Goal	To acquire the data for the WHOW platform.
Trigger	The data is provided to the WHOW platform.
Frequency	Not specified.
Preconditions	The data is available.
Minimal Post conditions	The WHOW platform has correctly processed the data.
Success Post conditions	Data is correctly made available on the WHOW platform.

Use Case ID	UC-0102
Title	Data transformation
Description	Cleaning, processing, and aggregating data by the WHOW platform.
Actor(s)	Data provider agent (human agent or software agent)
Goal	To process the data.
Trigger	The data is provided to the WHOW platform.
Frequency	Not specified.
Preconditions	The data is available.
Minimal Post conditions	The WHOW platform has correctly processed the data.

Success Post conditions	Data is correctly made available on the WHOW platform.

Use Case ID	UC-0103
Title	Data validation
Description	Data has to be validated for quality assessment purposes of the Knowledge Graph.
Actor(s)	Data provider agent (human agent or software agent)
Goal	Validate the data.
Trigger(s)	Data update or new data available
Frequency	Unknown - it can be done when data is updated or at specific time periods defined by the data provider (e.g., twice a year).
Preconditions	A validation specification exists and the (new) data is available
Minimal Post conditions	A validation report is produced.
Success Post conditions	Data is successfully validated.

Use Case ID	UC-0104
Title	Configuration of data update policies

Description	Data update processes have to be configured according to specific frequencies that depend on the data lifecycle.
Actor(s)	Data provider agent (human agent or software agent)
Goal	To set up all the components ruling the update of the data processes
Trigger(s)	Definition and change of frequency of data update
Frequency	On demand
Preconditions	A data update process has been designed.
Minimal Post conditions	Not specified.
Success Post conditions	Data and the related processes of update are configured with the correct frequency.

Use Case ID	UC-0105
Title	Metadata production
Description	When metadata is missing or has to be updated, it has to be handled by the WHOW platform.
Actor(s)	Data provider agent (human agent or software agent)
Goal	Production of metadata.
Trigger	Open datasets have been produced or modified.
Frequency	When needed, also according to the data frequency of update.
Preconditions	The open datasets should have been produced.

Minimal Post conditions	The WHOW platform has correctly processed or updated metadata.
Success Post conditions	Metadata is correctly made available on the WHOW platform and read by external platforms such as European Data Portal and national open data catalogue; that is, metadata is also fully compliant with European standards and related national extensions.

Use Case ID	UC-0106
Title	Metadata made available
Description	The data provider makes available metadata, along with the data, that has to be compliant to the standards.
Actor(s)	Data provider agent (human agent or software agent)
Goal	To make metadata available for the access by data consumers
Trigger	Metadata is provided to the WHOW platform.
Frequency	Not specified.
Preconditions	Metadata produced according to European/national standards.
Minimal Post conditions	The WHOW platform publishes a set of metadata.
Success Post conditions	Metadata is correctly made available on the WHOW platform and read by external platforms such as European Data Portal and national open data catalogue; that is, metadata is also fully compliant with European standards and related national extensions.

Use Case ID	UC-0107
Title	Ontology extension
Description	WHOW's ontology is unique and shared among all the nodes of the Knowledge Graph. When new data is acquired, possible extensions of the ontology (or network of ontologies) are to be considered.
Actor(s)	Data provider agent (human agent or software agent)
Goal	Extension of ontology in order to represent new requirements coming from new acquired data.
Trigger	Data is correctly acquired by the platform
Frequency	Not specified.
Preconditions	The ontology (or the network of ontologies) should have been previously created and published.
Minimal Post conditions	Not specified.
Success Post conditions	The ontology has been correctly extended to represent new requirements of the newly acquired data.

4.2.3 Requirements

For the data provision high-level requirement, we identified the following functional and non-functional requirements.

4.2.3.1 Functional Requirements

ID	Requirement	Use Case
FR-01	The platform shall support the definition of automated data cleaning, processing, transformation and publication pipelines involving datasets originating from the data providers' internal/existing systems and from external data sources.	UC-0101
FR-02	The platform shall be able to use data from other data providers than the ones already included in WHOW.	UC-0101
FR-03	The platform shall support the acquisition of heterogeneous data from different sources.	UC-0101
FR-0301	● The platform shall support the acquisition of data from non-RDF data sources available by means of structured open data formats, e.g. delimiter-separated values, JSON, XML (at the data provider site or from external providers).	UC-0101
FR-0302	● The platform shall be able to support the acquisition of data managed through relational database management systems (RDBMSs) and accessible via SQL.	UC-0101
FR-0303	● The platform shall support the acquisition of non-RDF external datasets accessible via RESTful APIs.	UC-0101
FR-0304	● The platform shall be able to support the acquisition of existing linked data managed through triplestore and made accessible via SPARQL.	UC-0101
FR-04	The platform shall include data storage capabilities for the physical storage and management of the knowledge graph	UC-0101
FR-0401	● Data shall be stored in the TripleStore.	UC-0101
FR-05	The TripleStore shall organise data in named graphs.	UC-0101

FR-06	The IRIs of entities in the knowledge graph shall address well defined production schemata, e.g. best practices for persistent IRIs[8].	UC-0102
FR-07	The platform shall enable the definition and management of data transformation/mapping rules for the processing of datasets in heterogeneous formats	
FR-0701	• The platform shall support the transformation of data from non-RDF data sources available by means of structured open data formats, e.g. delimiter-separated values, JSON, XML (at the data provider site or from external providers) into RDF.	UC-0102
FR-0702	• The platform shall support the transformation of non-RDF external datasets accessible via RESTful APIs.	UC-0102
FR-0703	• The platform shall support the transformation of data managed through relational database management systems (RDBMSs) and accessible via SQL	UC-0102
FR-08	The platform shall provide an extensible and pluggable set of data cleaning and transformation services. In the case specific privacy preserving operations are needed when the data is acquired in WHOW, the transformation services shall deal with them.	UC-0102
FR-09	The platform shall provide an extensible and pluggable set of data linking services.	UC-0102
FR-10	The platform shall produce RDF open datasets, compliant with the ontology network.	UC-0102
FR-11	The platform shall support the validation of data for quality assessment purposes of the Knowledge Graph (using languages such as SHACL[9], ShEx[10]).	UC-0103

[8] https://joinup.ec.europa.eu/sites/default/files/document/2013-02/D7.1.3%20-%20Study%20on%20persistent%20URIs.pdf.

[9] https://www.w3.org/TR/shacl/.

[10] https://www.w3.org/2001/sw/wiki/ShEx.

FR-1101	● The platform shall support the validation of metadata for quality assessment purposes of the Knowledge Graph (using languages such as SHACL, ShEx).	UC-0103
FR-1102	● The platform shall support the validation of linked data for quality assessment purposes of the Knowledge Graph (using languages such as SHACL, ShEx).	UC-0103
FR-12	The platform shall support an automatic data update process.	UC-0104
FR-13	The system shall support the update of the data according to the specific and different data frequencies of update based on data providers' needs.	UC-0104
FR-14	The platform shall enable the production and management of machine readable, standardised metadata in compliance with the reference metadata definition vocabularies.	UC-0105
FR-1401	● The platform shall enable the production and management of metadata in compliance with the DCAT Application profile for data portals in Europe (DCAT-AP).	
FR-1402	● The platform shall enable the production and management of metadata in compliance with the GeoDCAT-AP metadata profile for geospatial datasets.	UC-0105
FR-1403	● The platform shall enable the production and management of metadata in compliance with national extensions of the DCAT-AP and GeoDCAT-AP metadata definition profiles.	UC-0105
FR-15	The platform shall host and make available the standardised dataset metadata, to allow other data providers and external (open) data portals/catalogues to discover and harvest the published metadata.	UC-0106
FR-1501	● The platform shall enable the harvesting of metadata through access services and interfaces in compliance with the harvesting technical requirements/constraints defined	UC-0106

	for data suppliers by the European Data Portal and the National Data Portals.	
FR-16	The platform shall produce RDF-based metadata associated with the produced RDF datasets.	UC-0106
FR-17	The platform shall support data interoperability, also in the presence of new upcoming data and thus new data requirements, and the rules for generating semantic relationships/links between data items and entities within different linked data sources and datasets (cross-site datasets interlinking and linking to external datasets/knowledge graphs).	UC-0107

4.2.3.2 Non Functional Requirements

ID	Requirement	Use Case
NFR-01	The platform should be scalable in order to accommodate the joining of additional providers to the overall knowledge graph.	UC-0101
NFR-02	The platform shall use secure protocols (e.g., https).	UC-0101
NFR-03	The platform should publish data with a system capable of ensuring accessibility requirements where applicable.	UC-0101
NFR-04	The platform should rely on ontologies and vocabularies to enable a “common understanding of the data”, aligning them to existing and relevant standards.	UC-0102
NFR-05	IRIs have to be persistent in time.	UC-0102

4.3 RESTful-based data consumption over HTTP(s)

HLR ID	HLR-02
Name	RESTful-based data consumption over HTTP(s)
Description	From the user's point of view, WHOW's Knowledge Graph can be navigated online via Web according to different views, both from the perspective of Linked Open Data and of the ontology network. The WHOW platform also gives the possibility to choose what language to use for navigating the graph in.
Actor(s)	Data consumer agent

4.3.1 Examples and scenarios

Scenario 1
Description. Marta, an Italian student, wants to navigate WHOW's ontology network by means of the ontology design patterns involved. **Analysis.** As the user selects the knowledge lens of ontology design patterns, she will be able to get a custom view. She will also be able to select her preferred language for the navigation through the navigation menu.

Scenario 2
Description. Marco would like to explore the instance values for the state of bathing waters in Italy. He can do so through WHOW's linked open data network by means of a resource-based navigation.

Analysis. Such a view is loaded in the specific user interface after it has been selected and can be retrieved either by giving the IRI of the information Marco would like to get more insight about or by step-by-step user exploration starting from the dataset.

4.3.2 Use Cases

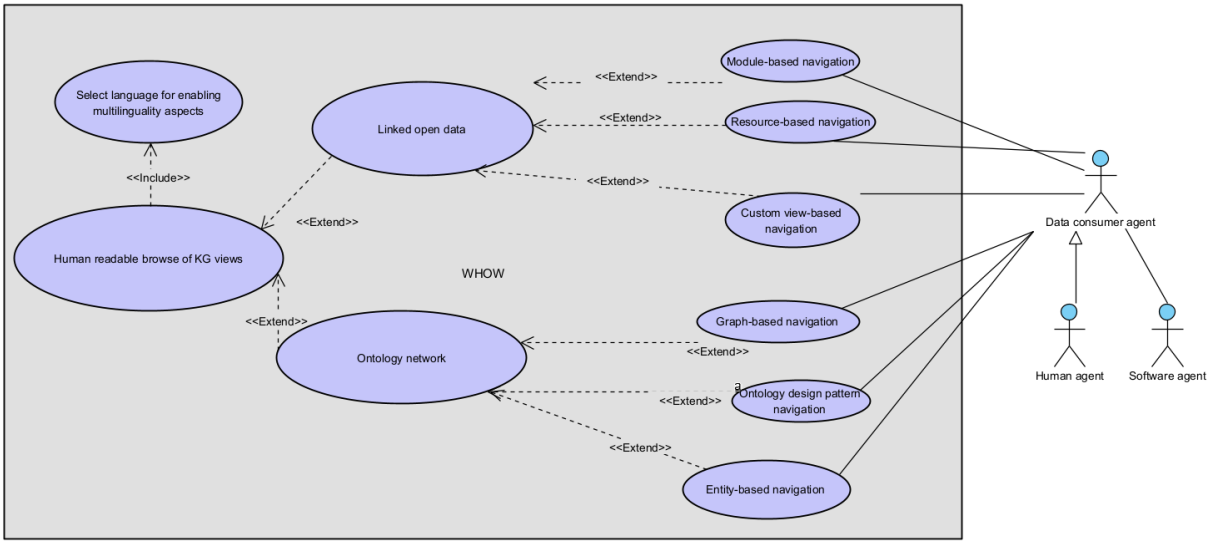

Figure 3: Use case diagram for HLR-02.

UC-0201 Linked open data—module-based navigation

UC-0202 Linked open data—resource-based navigation

UC-0203 Linked open data—custom navigation

UC-0204 Ontology network—entity-based navigation

UC-0205 Ontology network—graph-based navigation

UC-0206 Ontology network—ontology design pattern

UC-0207 Select language for enabling multilinguality aspects

Use Case ID	UC-0201
Title	Linked open data—module-based navigation
Description	Linked open data can be navigated through pattern-specific modules.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to navigate linked open data through modules.
Trigger	The agent selects the navigation they want to use.
Frequency	Not specified.
Preconditions	The view based on the pattern-specific module can be selected .
Minimal Post conditions	The view is loaded on the UI.
Success Post conditions	The view can be correctly navigated by the user.

Use Case ID	UC-0202
Title	Linked open data—resource-based navigation
Description	Linked open data can be navigated resource-wise; it is possible to view the overall dataset or the single resource.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to navigate linked open data through resources.

Trigger	The agents select the navigation they want to use.
Frequency	Not specified.
Preconditions	The view based on the specific resource can be selected.
Minimal Post conditions	The view is loaded on the UI.
Success Post conditions	The view can be correctly navigated by the agent.

Use Case ID	UC-0203
Title	Linked open data—custom navigation
Description	Linked open data can be navigated through a custom, user-friendly navigation.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to navigate linked open data through a custom view.
Trigger	The agents select the navigation they want to use.
Frequency	Not specified.
Preconditions	The custom view can be selected.
Minimal Post conditions	The view is loaded on the UI.
Success Post conditions	The view can be correctly navigated by the agent.

Use Case ID	UC-0204
Title	Ontology network—entity-based navigation
Description	The ontology network can be navigated through an entity-based view.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to navigate the ontology network through an entity-based view.
Trigger	The agents select the navigation they want to use.
Frequency	Not specified.
Preconditions	The view based on the ontology network's entities can be selected.
Minimal Post conditions	The view is loaded on the UI.
Success Post conditions	The view can be correctly navigated by the agent.

Use Case ID	UC-0205
Title	Ontology network graph-based navigation
Description	The ontology network can be visualized and searched graph-wise.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to navigate the ontology network through a graph-based view.
Trigger	The agents select the navigation they want to use.

Frequency	Not specified.
Preconditions	The graph-based view can be selected.
Minimal Post conditions	The view is loaded on the UI.
Success Post conditions	The view can be correctly navigated by the agents.

Use Case ID	UC-0206
Title	Ontology network—ontology design pattern
Description	The ontology network can be navigated through ontology design patterns.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to navigate the ontology network through ontology design patterns.
Trigger	The agents select the navigation they want to use.
Frequency	Not specified.
Preconditions	The view can be selected based on an ontology design pattern.
Minimal Post conditions	The view is loaded on the UI.
Success Post conditions	The view can correctly be navigated by the agents.

Use Case ID	UC-0207
Title	Select language for enabling multilinguality aspects
Description	The agent can select the language to enable multilinguality aspects.
Actor(s)	Agent (human agent or software agent)
Goal	To be able to select the language for the navigation.
Trigger	The agent selects the language.
Frequency	Not specified.
Preconditions	The multilingual aspects have been included in the ontologies and linked open data.
Minimal Post conditions	Not specified.
Success Post conditions	The navigation is in the selected language.

4.3.3 Requirements

For the data consumption high level requirement, we identified the following functional and non-functional requirements.

4.3.3.1 Functional requirements

ID	Requirement	Use Case
FR-18	The platform shall visualise information based on a provided lens that organises and aggregates data according to a given view.	UC-0201
FR-1801	● The platform shall provide a pattern-based visualisation of data based on a provided lens that organises and aggregates data according to a given view.	UC-0201
FR-19	The platform shall provide one with a number of filters based on the knowledge modelled by the ontology network and its modules, e.g. for enabling faceted browsing capabilities..	UC-0201
FR-20	The user shall be able to explore the Knowledge Graph incrementally by means of visual metaphors.	UC-0201
FR-21	The user shall be able to retrieve information about a resource by giving its IRI.	UC-0202
FR-22	The user shall be able to visualise the instance values related to a class, and properties of the Knowledge Graph.	UC-0202
FR-23	The user shall be able to visualize type-specific properties of an instance.	UC-0202
FR-24	The user shall be able to filter out and get details of the data according to specific users' requests.	UC-0203
FR-25	The user shall be able to visualise a human-readable HTML documentation of the ontologies part of the Knowledge Graph.	UC-0204
FR-26	The user shall be able to visualise the information related to a class and/or a property of the ontology network.	UC-0204

FR-27	The user shall be able to switch the visualisation and the navigation from Knowledge Graph instances to entities of the ontology network on demand during the exploration.	UC-0204
FR-28	The user shall be able to perform a panning function (moving the complete visualization around by dragging it or the background) on some elements of the ontology network.	UC-0204
FR-29	The user should be able to perform a graphical zoom on the ontology network in order to gather more details on some of its components being either ontologies or ontological entities.	UC-0205
FR-30	The user shall be able to search and highlight specific parts of the ontology network graph view.	UC-0205
FR-31	The user shall be able to filter out parts of the ontology network graph view.	UC-0205
FR-32	The user shall be able to perform custom editing on the ontology network graph view.	UC-0205
FR-33	The user shall be able to navigate the ontology network by means of ontology design patterns used as knowledge lenses.	UC-0206
FR-34	The user shall be able to select his/her preferred language for ease of navigation of the Knowledge Graph.	UC-0207
FR-35	The platform shall support content negotiation.	UC-[0201,UC-0207][11]
FR-36	All the resources shall be dereferenced by https.	UC-[0201,UC-0207]

[11] The notation in this case means that the use cases that have been addressed by the requirement are all those encompassed by the specified range, e.g. UC-[0201,UC-0207] encompasses the UCs ranging from 0201 to 0207.

4.3.3.2 Non Functional requirements

For the RESTful-based data consumption over HTTP(s) high-level requirement, no non-functional requirements have been identified.

4.4 Query-based data consumption

HLR ID	HLR-03
Name	Query-based data consumption
Description	The Knowledge Graph can be queried in multiple way:, with defined SPARQL standard protocol and language, its defined extensions and REST API paradigm, by both human and software agents. Furthermore, this includes a machine-to-machine interaction scenario, in order to provide users with predefined or customized selection of data.
Actor(s)	Agent (human or software agent)

4.4.1 Examples and scenarios

Scenario 1
Description. Giada, a data journalist who is not proficient with SPARQL standard protocol and language, wants to know how many accesses to emergency rooms in Lombardy have happened during the year 2020 and if there could be a correlation between pathogens found in the region's drinking waters and spike in accesses in the hospital for gastrointestinal diseases in specific areas. **Analysis.** Although Giada does not know how to use SPARQL, her need can be aided by a predefined guided query available on the WHOW website. The availability of a predefined query allows Giada to run it to get the possible results or to understand how to query data so that she can modify it according to her needs within the editor.

Scenario 2
Description. Roberto, a computer scientist, wants to set up a public dashboard of data harvested from the WHOW Knowledge Graph, with the aim of tracking the quality of marine waters in Italy over time. For this, he needs to use a REST API to do so and have the data updated automatically. **Analysis.** Roberto does not need to manually retrieve and download the data concerning his research. Using the WHOW API, he is able to automatically request data from existing records and ensure that it is up-to-date. Any possible programming language able to support a REST client is supported (e.g. Python, Java) and there is no limit to the data Roberto can access. The API returns output in different structured formats, including some of the most adopted, such as JSON-based (e.g. JSON-LD) and CSV-formatted data.

4.4.2 Use cases

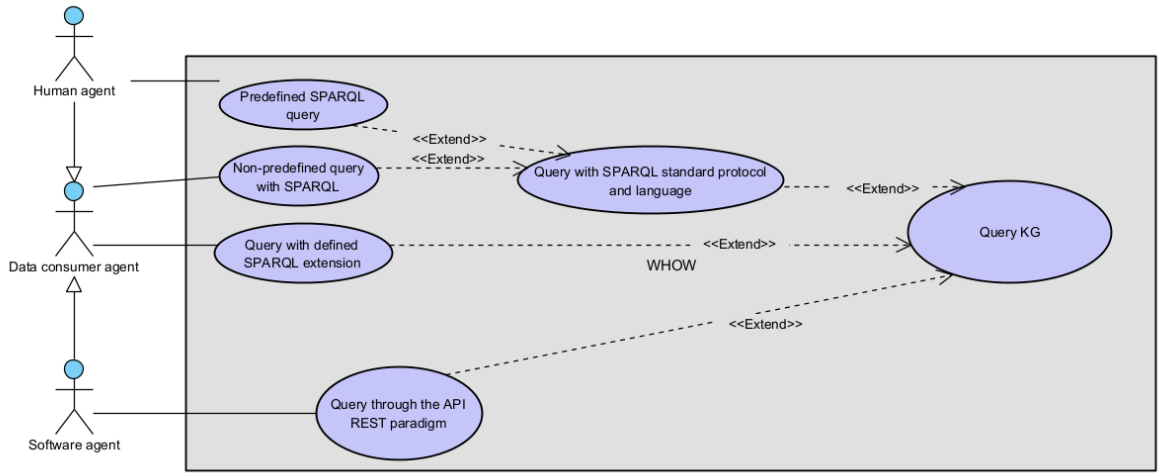

Figure 4: Use case diagram for HLR-03.

UC-0301 Query with defined SPARQL extension

UC-0302 Non-predefined query with SPARQL standard protocol and language

UC-0303 Predefined SPARQL query

UC-0304 Query through the REST API paradigm

Use Case ID	UC-0301
Title	Query with defined SPARQL extension
Description	The agent can perform a query through a defined SPARQL extension (e.g., GeoSPARQL).
Actor(s)	Agent (human agent or software agent)
Goal	To perform a query with a defined SPARQL extension.
Trigger	The agent has issued a SPARQL query using a SPARQL extension.
Frequency	On demand.
Preconditions	The syntax of the query is correct.
Minimal Post conditions	The query is successfully performed.
Success Post conditions	Results are correctly returned to the agent.

Use Case ID	UC-0302
Title	Non-predefined query with SPARQL standard protocol and language

Description	The agent can run a non-predefined query through SPARQL.
Actor(s)	Agent (human agent).
Goal	To perform a custom query with SPARQL.
Trigger	The agent has issued a SPARQL query.
Frequency	On demand.
Preconditions	The syntax of the query is correct.
Minimal Post conditions	The query is successfully performed.
Success Post conditions	Results are correctly returned to the agent.

Use Case ID	UC-0303
Title	Predefined SPARQL query
Description	The agent can perform a predefined query through SPARQL.
Actor(s)	Agent (human agent).
Goal	To perform a predefined query with SPARQL.
Trigger	The query is selected by the agent.
Frequency	On demand.
Preconditions	A set of predefined SPARQL queries are defined.

Minimal Post conditions	The query is successfully performed.
Success Post conditions	Results are correctly returned to the agent.

Use Case ID	UC-0304
Title	Query through the REST API paradigm.
Description	The software agent can perform a query through the REST API paradigm.
Actor(s)	Agent (Software agent)
Goal	To perform a query through the REST API paradigm.
Trigger	The REST API is well-documented and available.
Frequency	On demand.
Preconditions	The REST API documentation is available.
Minimal Post conditions	Not specified.
Success Post conditions	The results are returned to the software agent.

4.4.3 Requirements

For the query-based data consumption of the Knowledge Graph high level requirement, we identified the following functional and non-functional requirements.

4.4.3.1 Functional requirements

ID	Requirement	Use Case
FR-37	An agent shall be able to execute (remote) queries on the Knowledge Graph in SPARQL and its supported extensions (e.g. GeoSPARQL).	UC-0301
FR-3701	● The platform shall provide an access service that is able to support SPARQL protocol requests and possibly SPARQL protocol extensions such as GeoSPARQL.	UC-0301
FR-38	The platform shall provide the user with a UI for querying the Knowledge Graph.	UC-0302
FR-39	The platform shall support the execution of federated queries over the knowledge graph and across multiple sites (cross-site queries and queries involving external endpoints).	UC-0302
FR-40	The platform shall provide access to the open Knowledge Graph available at each site via one or more SPARQL endpoints.	UC-0302
FR-41	The human agent shall be able to perform custom queries on the Knowledge Graph via a human interaction service.	UC-0302
FR-42	The human agent shall be able to select a predefined SPARQL query from a set of example queries provided through a UI.	UC-0303
FR-43	The platform shall expose a set of REST APIs, compliant with the ontology network, for data access and consumption, documented in compliance with the OpenAPI Specification (OAS).	UC-0304

4.4.3.2 Non Functional requirements

For the Query-based data consumption high-level requirement, no non-functional requirements have been identified.

4.5 Download/export of the knowledge graph

HLR ID	HLR-04
Name	Download/export Knowledge Graph both in an open data format or with RDF serializations or formats.
Description	The requirement assesses the possibility to export the Knowledge Graph.
Actor(s)	Data consumer agent

4.5.1 Examples and scenarios

Scenario 1
Description. Francesca, a researcher, would like to download some WHOW datasets for the past two years to see if it is possible to find a correlation between drinking water in her city and infectious diseases. She needs the data in an open format, so that she can easily analyze it and make her analyses repeatable by other scholars when the paper is published. **Analysis.** WHOW grants the download of data in an open data format, which can be XML, JSON, delimiter-separated values formats (CSV, TSV and similar formats for tabular data), RDF. After selecting one of the formats, Francesca can successfully download and save the file on her device. Data from the data provider is already cleaned, transformed, and aggregated as per HLR-01.

4.5.2 Use Cases

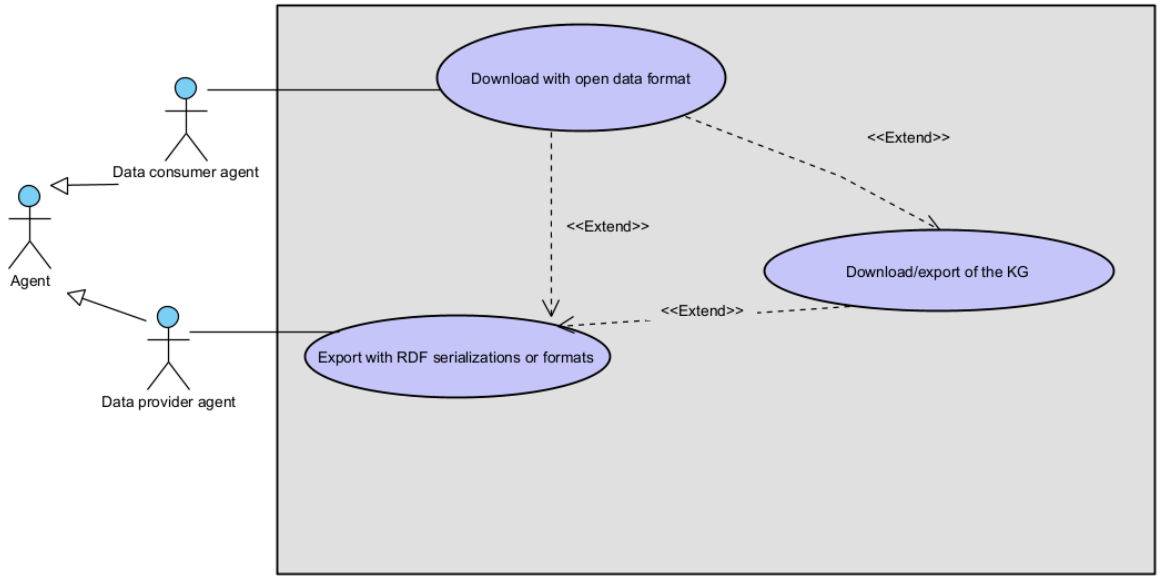

Figure 5: Use case diagram for HLR-04.

UC-0401 Download with open data format

UC-0402 Export with RDF serializations or formats

Use Case ID	UC-0401
Title	Download with open data format
Description	The platform makes it possible to download the data in an open data format.
Actor(s)	Agent(human agent)
Goal	To download the Knowledge Graph.
Trigger	The preferred format for download is selected by the human agent.

Frequency	On demand.
Preconditions	The preferred format can correctly be selected.
Minimal Post conditions	The file is downloaded.
Success Post conditions	The file is downloaded successfully and can be opened properly.

Use Case ID	UC-0402
Title	Export with RDF serializations or formats
Description	The data provider can export the data in RDF serialisations or formats.
Actor(s)	Data provider
Goal	To export the data in a chosen RDF serialisation or format.
Trigger	The data provider selects the RDF serialisation or format for exporting the data.
Frequency	On demand.
Preconditions	The export RDF serialisation or format can be chosen.
Minimal Post conditions	The data is exported.
Success Post conditions	The file is exported successfully and can be opened properly.

4.5.3 Requirements

For the Download/export of the knowledge graph high-level requirement, we identified the following functional and non-functional requirements.

4.5.3.1 Functional requirements

ID	Requirement	Use Case
FR-44	The user shall be able to download RDF data as dump files in RDF open format, e.g. RDF turtle, RDF-XML, JSON-LD.	UC-0401
FR-45	The platform shall support downloading in the following open data formats: XML, JSON, delimiter-separated values formats (CSV, TSV and similar formats for tabular data),	UC-0401
FR-46	The data provider shall be able to export his/her data in different RDF serialisations or formats.	UC-0402

4.5.3.2 Non Functional requirements

ID	Requirement	Use Case
NFR-06	The platform should be able to load, on a SPARQL endpoint, files of potentially big dimensions.	UC-0401

4.6 Ontology development and maintenance

HLR ID	HLR-05
Name	Ontology development and maintenance

Description	Processes for ontology development using eXtreme Design with Content Ontology Design Patterns (XD) are described. Additionally, versioning policies are defined for maintenance of the semantic assets being created.
Actor(s)	Agent (ontology engineer)

4.6.1 Examples and scenarios

Scenario 1
Description. An ontology engineer wants to design and implement data model schemes for the hydrogeological basins in WHOW. He/she is required to use eXtreme Design with Content Ontology Design Patterns (XD) and thus interact with domain experts over regular meetings in order to list out all the needed requirements and minimize the number of assumptions to be made. **Analysis.** After having made sure together with the domain experts there are no untreated requirement stories, the ontology engineers initiate ontology modules modelling by searching for Ontology Design Patterns to reuse. They find there are existing ontology design patterns (such as observation-measure design pattern, classification design pattern, spatial objects and geometry design pattern, pollution design pattern), but there are ontology modules to be modelled from scratch. If, after testing, all competency questions are satisfied, such additional models are integrated into the final ontology, aligned with other ontologies and then test once again such integration. The process is incremental, as shown by Figure 9.

4.6.2 Use cases

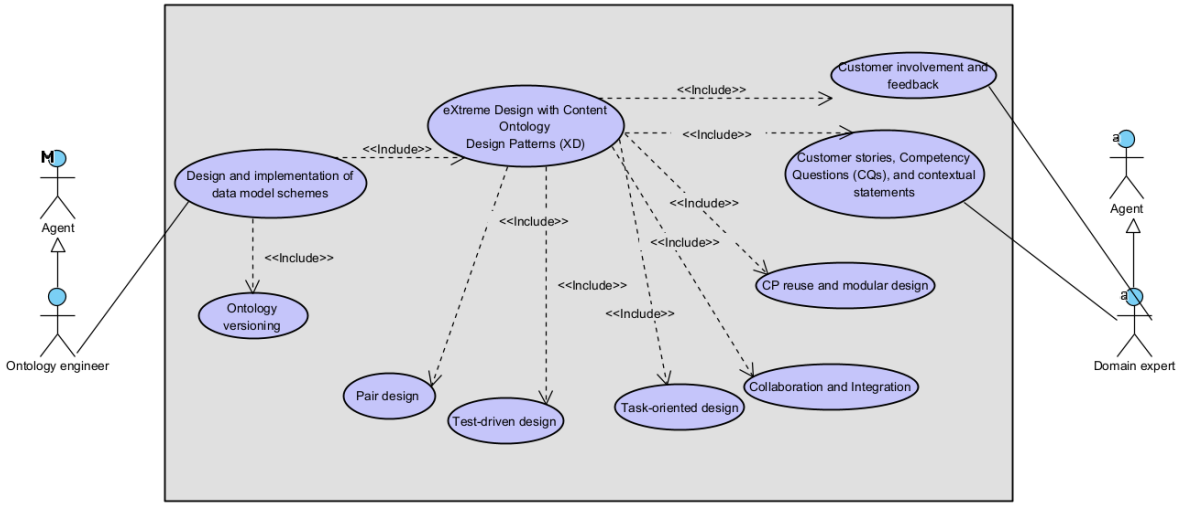

Figure 6: Use case diagram for HLR-05.

UC-0501 Design and implementation of data model schemes

UC-0502 Reuse and extension of existing ontology design patterns

UC-0503 Customer involvement and feedback

UC-0504 Customer stories, Competency Questions (CQs), and contextual statements

UC-0505 CP reuse and modular design

UC-0506 Collaboration and integration

UC-0507 Task-oriented design

UC-0508 Test-driven design

UC-0509 Ontology Versioning

Use Case ID	UC-0501
Title	Design and implementation of data model schemes
Description	The ontology engineer is meant to design and implement ontologies.

Actor(s)	Agent (ontology engineer)
Goal	To design and implement data model schemes.
Trigger	Availability of competency questions, also coming from domain experts
Frequency	Not specified.
Preconditions	A preliminary analysis of the domain has been carried out. Pairs for working are designated; domain experts are available to be consulted during the design process.
Minimal Post conditions	Data model schemes are designed.
Success Post conditions	The data model schemes have been successfully developed and answer all the competency questions.

Use Case ID	UC-0502
Title	Reuse and extension of existing ontology design patterns
Description	Ontology engineers design the ontology using the eXtreme Design with Content Ontology Design Patterns (XD). It is a collaborative, incremental, iterative method for pattern-based ontology design.
Actor(s)	Agent (ontology engineer)
Goal	To use XD methodology to design the network of ontologies for the project.
Trigger	Not specified.
Frequency	Not specified.
Preconditions	Availability of a catalogue of ontology design patterns to be reused and/or extended.

Minimal Post conditions	Not specified.
Success Post conditions	The XD methodology has been successfully adopted with ontology design patterns reused and properly extended as required.

Use Case ID	UC-0503
Title	Customer involvement and feedback
Description	The customer is involved in the ontology development and its representative should be a team, whose members are aware of all parts and needs of the project.
Actor(s)	Agent (ontology engineer, domain expert)
Goal	The customer contributes and is involved in the ontology development.
Trigger	Not specified.
Frequency	Not specified.
Preconditions	The customer, along with its representative, is available and aware of all parts and needs of the project.
Minimal Post conditions	Both the customer and its representative are successfully involved in the design process and are correctly aware of what tasks the application is expected to solve.
Success Post conditions	The possible number of assumptions that the ontology engineers have to make on the incomplete requirement descriptions is minimized.

Use Case ID	UC-0504
Title	Customer stories, Competency Questions (CQs), and contextual statements
Description	The ontology requirements and its tasks are described in terms of small stories by the customer representative, which are then transformed in the form of CQs and contextual statements by both ontology engineers and domain experts.
Actor(s)	Agent (ontology engineer, domain expert)
Goal	To have the ontology requirements described by customer stories, competency questions and contextual statements.
Trigger	Not specified.
Frequency	Not specified.
Preconditions	The actors involved in this use case are available to work on customer stories, CQs and contextual statements.
Minimal Post conditions	The customer successfully helped in making explicit as much implicit knowledge as possible.
Success Post conditions	Customer stories, CQs and contextual statements are successfully outlined.

Use Case ID	UC-0505
Title	Modular design
Description	The goal of modular design is to create ontological modules according to some policies
Actor(s)	Agent (ontology engineer)

Goal	To modularize ontologies for maintenance purposes.
Trigger	Not specified.
Frequency	Not specified.
Preconditions	Availability of data model schemes.
Minimal Post conditions	Not specified.
Success Post conditions	Ontological modules are created.

Use Case ID	UC-0506
Title	Task-oriented design
Description	A design approach focused on the tasks that the ontology is expected to address.
Actor(s)	Agent (ontology engineer)
Goal	To focus on a task-oriented design rather than a philosophical approach.
Trigger	Not specified.
Frequency	Not specified.
Preconditions	The tasks are to be known.
Minimal Post conditions	Not specified.

Success Post conditions	Task-oriented design is employed by the ontology engineer(s).

Use Case ID	UC-0507
Title	Test-driven design
Description	Stories, CQs, and contextual statements are used in order to develop unit tests.
Actor(s)	Agent (ontology engineer)
Goal	To successfully develop unit tests for addressing certain user stories associated with certain competency questions.
Trigger	Ontology engineers run the SPARQL queries.
Frequency	Not specified.
Preconditions	SPARQL queries that formally encode the competency questions are defined and associated with the expected results.
Minimal Post conditions	The SPARQL queries have run successfully.
Success Post conditions	All unit tests on one or more user stories have successfully passed.

Use Case ID	UC-0508
Title	Ontology versioning
Description	This use case relates to the identification and management of the different

	versions of an ontology.
Actor(s)	Agent (ontology engineer)
Goal	To identify and manage the different versions of an ontology.
Trigger	Major changes in the ontology design are applied.
Frequency	Not specified.
Preconditions	An ontology network exists and changes of ontologies involved in the Knowledge Graph have been requested.
Minimal Post conditions	Not specified.
Success Post conditions	Different versions of the ontologies involved in the Knowledge Graph are successfully applied and handled.

4.6.3 Requirements

For the ontology development and maintenance high level requirement we identified the following functional and non-functional requirements.

4.6.3.1 Functional requirements

ID	Requirement	Use Case
FR-47	The WHOW platform should provide functionalities to identify and manage changes in the ontologies of the ontology network.	UC-0509

4.6.3.2 Non functional requirements

ID	Requirement	Use Case
NFR-07	The design and implementation of data model schemas should follow eXtreme Design with Content Ontology Design Patterns (XD) strategy.	UC-0501
NFR-08	The domain expert should be consulted in order for the ontology design to define the expected tasks of the application and ease the explicit expression of knowledge.	UC-0502
NFR-09	The ontology engineer should reuse relevant Content Patterns (CP) to cover the requirements.	UC-0505
NFR-10	Local competency questions should match those covered by a CP in order for the ontology engineer to evaluate the CP's suitability for solving the local problems.	UC-0505

5. Linked Open Data reference architecture

Based on the use cases and related functional and non-functional requirements detailed in Section 3, this section presents the design of the Linked Open Data reference architecture that can be deployed by all those data providers that are willing to join the WHOW knowledge graph. Specifically, we envision the creation of knowledge graphs on water pollution and consumption, and health parameters and disease dissemination at each data provider. These knowledge graphs are ultimately federated all together thanks to native semantic links defined in the datasets and enabled through the use of semantic web open standards (e.g., RDF - Resource Description Framework). The knowledge graphs can be accessed by anyone through an ecosystem of pluggable services that contribute to the construction of the overall Linked Open Data reference architecture (also named WHOW toolkit). In essence, a decentralization approach is employed by WHOW, also making the overall toolkit in line with recent discussions and business specifications like Solid [4]. Data and applications can be fully decoupled, breaking data silos, ensuring a better control on data and its privacy when required by rights holders, and fostering the deployment of more sustainable data management processes.

A high-level architecture overview is first illustrated and details of each architectural layer are then described, also using the UML modelling language as chosen formalism for their design.

5.2 High level architecture overview

The high-level reference Linked (Open) Data architecture is depicted in Figure 8.

It is a fully distributed architecture that allows different data providers, also from various European member states, to build, manage over time and publish their own decentralised knowledge graphs, which together form the so-called WHOW knowledge graph, along with associated services.

The architecture, to be deployed by each data provider, is designed as a five-layer architecture consisting of the:

- *data preparation layer*, responsible for extracting knowledge both from the internal sources of the WHOW project data providers and from external sources managed by other stakeholders, not directly participating in WHOW. All data sources can potentially be very heterogeneous both in formats and structures;

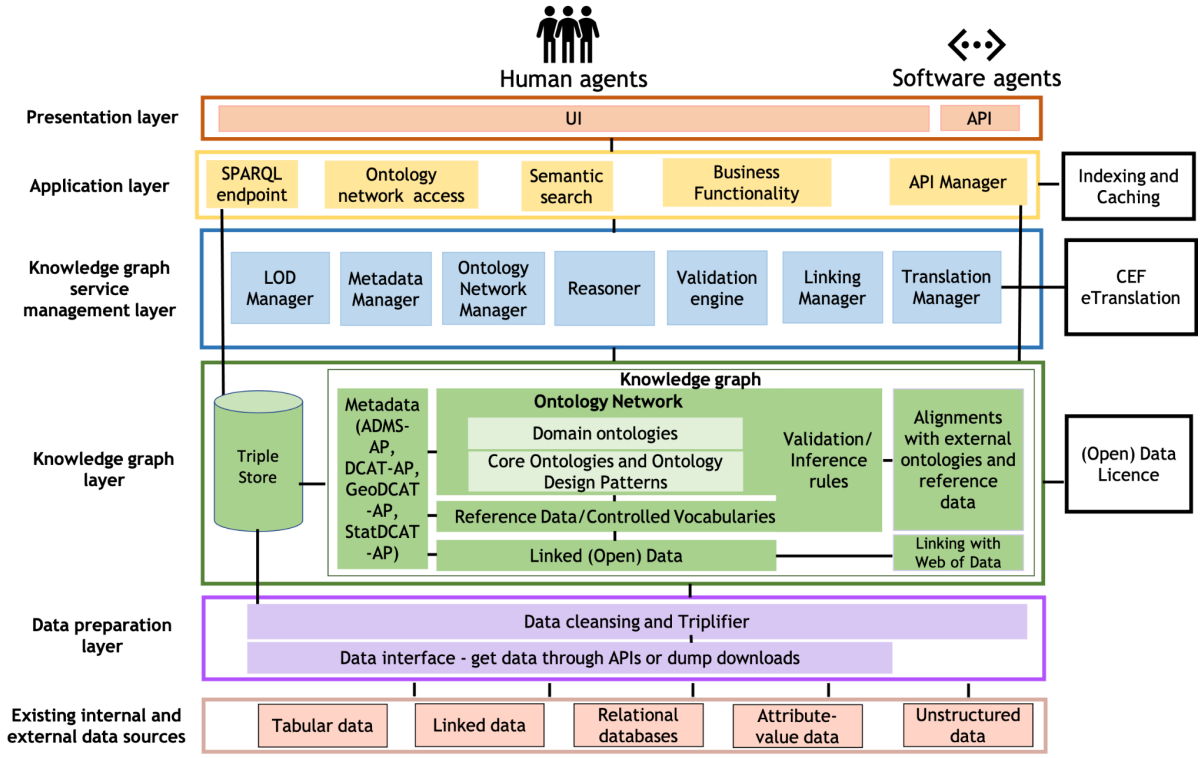

Figure 7: WHOW high level architecture

- *knowledge graph layer* which includes the domain-dependent semantic assets to be produced and published. In particular, the main components of this layer are both shared ontologies and reference data (i.e., controlled vocabularies), used to model the WHOW knowledge, and data represented using those ontologies and reference data, published in the form of Linked Open Data (LOD). All these semantic assets are expressed using open semantic web open standards and formats (e.g., OWL, RDF), and are associated with an open standard license (e.g., Creative Commons Attribution 4.0 - CC-BY 4.0, Creative Commons Public Domain - CC0) so as to maximize their re-use by anyone;.
- k*nowledge graph service management layer*, responsible for providing the set of software components that are used to manage the underlying knowledge graph layer. This layer consists of all those services that allow the actors, identified in the earlier Section 3, to manage the different elements of the knowledge graph, from metadata, network of ontologies up to the linked open data, with a focus on the multilingual and semantic linking aspects of this latter. It is in this layer that:
    - (Geo)DCAT-AP metadata, also compliant with national extensions, is properly managed. This allows data providers to make their knowledge graph resources harvestable by the European Data portal (data.europa.eu) and other catalogues of reference (e.g., national open data catalogues, European geospatial data catalogue);
    - the lifecycle of the network of ontologies and reference data is managed, spanning from versioning management, necessary for capturing the possible changes over

time of the semantic assets, to semantic inference and consistency checks, required as a form of verification of the robustness of the produced semantic assets;
    - linked open data management activities are performed. In fact, this layer includes services for data validation. against ontologies and reference data, data multilinguality, through the use of CEF building blocks such as e-Translation, data linking to create semantic links among different datasets, and any other CRUD operations on the linked open data;
- *application layer and presentation layer,* responsible for supporting all those use cases that have been identified related to the consumption by end-users and software agents of the knowledge graph. Therefore, these layers embody all those components that enable both a human-based (UI of the presentation layer in Figure 7) and a machine-based interaction (API of the presentation layer in Figure 7), where searching and accessing functionalities are provided. SPARQL endpoint instances for data and ontology network access are part of this layer as well as all those software components that will allow WHOW data providers to offer APIs defined via REST, using standards such as OpenAPI[12]. The application layer may also include specific business dependent functionalities.

The lines in Figure 7 represent communications among the layers, and thus among their components.

It is worth noting that the architecture has been designed to be independent of the specific water and health application domains of WHOW. With the exception of the knowledge graph layer where inevitably the elements are strongly dependent on the characteristics of these domains, the other layers and services of the architecture are sufficiently general to be applied in any Linked Open Data technical scenario, making the proposed architecture a so-called r*eference architecture*.

In addition, some services may be viewed as mandatory in order to publish Linked Open Data (i.e., SPARQL endpoint, Triplifier): without them there is no possibility to create and publish a knowledge graph; other services can be thought of as additional services whose aim is to contribute to the creation of a more comprehensive toolkit where also all the aspects of data quality are taken into account (examples include the validation service, semantic search, indexing/caching).

In Table 1 we introduce the mapping between the architectural layers and the High Level Requirements (HLRs) previously presented in this deliverable. The following subsections describe in more details each layer with the related components.

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

Layer	HLR
Presentation layer	HLR-02 RESTful-based data consumption over HTTP(s) HLR-03 Query-based data consumption
Application layer	HLR-02 RESTful-based data consumption over HTTP(s) HLR-03 Query-based data consumption HLR-04 Download/export Knowledge Graph
Knowledge graph service management layer	HLR-01 Data provision HLR-05 Ontology development and maintenance
Knowledge graph layer	HLR-01 Data provision HLR-04 Download/export Knowledge Graph HLR-05 Ontology development and maintenance
Data preparation layer	HLR-01 Data provision HLR-04 Download/export Knowledge Graph

Table 1: Mapping between architectural layers and HLRs.

We remind the reader that a component diagram depicts how components are wired together to form larger components or software systems. They are typically used to illustrate the structure of arbitrarily complex systems. The graphical notation used distinguished between (i) components, which are represented as boxes, (ii) provided interfaces, which are represented as solid lines starting from a component and ending with a solid circle, and (iii) required interfaces, which are represented as solid lines starting from a component and ending with a empty half-circle. The colours used for depicting components and interfaces in this diagram, as well as in subsequent diagrams, depend on those used in Figure 7. Accordingly, we use violet for the data preparation layer, blue for the knowledge graph service management layer, etc.

5.3 Architectural style and deployment

The design relies on the component-based and REST architectural styles. The component-based architectural style describes a software engineering approach to system design and development. It focuses on the decomposition of the design into individual functional or logical components that expose well-defined communication interfaces containing methods, events, and properties. This provides a high level of abstraction, even higher than object-oriented design principles, and does not

focus on issues such as communication protocols and shared states. The key principle of the component-based style is the use of components that are software packages, web services, web resources, or modules that encapsulate a set of related functions and data. In the component-based style, each component addresses the following properties [17]:

- **Reusability.** Components are usually designed to be reused in different scenarios in different applications. However, some components may be designed for a specific task.
- **Replaceability.** Components may be readily substituted with other similar components.
- **Non context specificity.** Components are designed to operate in different environments and contexts. Specific information, such as state data, should be passed to the component instead of being included in or accessed by the component.
- **Extensibility.** A component can be extended from existing components to provide new behavior.
- **Encapsulability.** Components expose interfaces that allow the caller to use its functionality, and do not reveal details of the internal processes or any internal variables or state.
- **Independence.** Components are designed to have minimal dependencies on other components. Therefore components can be deployed into any appropri- ate environment without affecting other components or systems.

The main benefit of designing software by adopting the Component-based architectural style derives from the principle of encapsulation. In fact, each component exposes its functionalities to the rest of the system by providing an interface, which specifies the services available and hides implementation details to other components. Hence, with regard to system-wide coordination, components communicate with each other via interfaces. This makes it easy to add new components, to substitute them and to modify the configuration of the communication among components, which means to make customisable the system behaviour.

The Representational State Transfer [18] (REST) architectural style defines a set of constraints about how the architecture can be designed in a distributed fashion over hypermedia systems, such as the Web. The REST architectural style mainly focuses on the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems.

Accordingly, the architecture can be instantiated and deployed into distributed nodes that form the WHOW infrastructure and communicate over the infrastructure by means of RESTful services. Each node does not need to deploy the whole architecture, but it might instantiate some of the components that are required according to specific needs at node level. For example, a node might instantiate the components of the *Data preparation layer* and communicate via REST over the Web with another node that instantiates the components composing the *Knowledge graph layer*.

5.4 Data preparation layer

The general goal of WHOW is to enable the creation of a knowledge graph that represents data about water quality and health parameters coming from heterogeneous sources and open data formats that are currently a knowledge soup that can be hardly accessed, queried, interpreted and managed homogeneously. Accordingly, we envision the data preparation layer as the set of software components that provide the WHOW toolkit with data transformation capabilities. In this context, by *transformation* we mean the process of converting heterogeneous open data formats coming from a variety of data providers (internal or external) to RDF. Figure 8 shows the UML component diagram of the data preparation layer.

5.4.1 Layer's software components

In Figure 8 we model the data preparation layer as composed of the following components and their corresponding interfaces:

- **OpenDataAcquirer:** the OpenDataAcquirer provides functionalities that enable a data provider (being either a human or an artificial agent) to submit data to WHOW. Hence, data are acquired by WHOW and they can be provided in any possible open data format supported by the framework (e.g. CSV, TSV, JSON, XML, etc.). The interaction between the data provider and the component is enabled by the interface labelled as IDataAcquisition provided by the component itself. Once data is acquired then it is possible to (i) populate an open data catalogue managed by WHOW with relevant metadata and (ii) store the acquired data. The responsibles for those operations are the components named OpenDataCataloguer and InternalOpenDataStorageManager, which are linked to the OpenDataAcquirer by means of the interfaces IDataCataloguing and IStorage, respectively.
- **OpenDataCataloguer:** the OpenDataCataloguer is responsible for data cataloguing. This is performed by using well known cataloguing standards, such as the Data Catalog Vocabulary[13] (DCAT). DCAT is an RDF vocabulary designed to facilitate interoperability between data catalogs published on the Web. The OpenDataCataloguer aims at increasing discoverability and enabling other components to consume metadata. Metadata can be provided to the OpenDataCataloguer through the interface named IDataCataloguing. Similarly, metadata can be queried and retrieved through the interface named IMetaDataRetrieval. Both interfaces are provided by the OpenDataCataloguer. For instance, the OpenDataAcquirer submits to OpenDataCataloguer the metadata of a dataset that has been just acquired by using the interface IDataCataloguing. The OpenDataCataloguer enables the harvesting of metadata through access services and interfaces in compliance with the harvesting technical

[13] https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/dcat-application-profile-data-portals-europe/release/11

requirements/constraints defined for data suppliers by the European Data Portal and the National Data Portals.

- **InternalOpenDataStorageManager:** the InternalOpenDataStorageManager is the component of the framework that enables the storage of open datasets uploaded by data providers into an internal repository. The data coming from data providers are not modified for internal storing, that is, no transformation is applied and they are kept in the original form. On one hand, the storing capability is exposed by the component by means of the IStorage interface the latter provides to the other components. On the other hand, the interface IDataStoreAccess is provided by the component for enabling other components to access stored data.
- **Transformer:** the transformer is meant as the component of the layer aimed at performing the generation of RDF data from original sources. The transformation solution implemented by the component is modular. In fact, it separates the syntactic transformation (i.e. syntactic reengineering) from the semantic enhancement (i.e. semantic refactoring) of the original sources. Syntactic reengineering is concerned with a mere syntactic transformation of the original source to RDF without addressing any specific domain semantics. Accordingly, the resulting RDF is typically modelled according to a vocabulary that describes the format of the original data source, e.g. a vocabulary that allows one to describe as RDF CSV data in terms of their rows, columns, and cells. Domain semantics is then achieved by means of semantic refactoring that re-organise RDF data according to peculiar ontologies or vocabularies. This will allow WHOW to address the heterogeneity of formats (e.g. CSV, JSON, XML, etc.) and hidden semantics that are inherently associated with non-RDF sources flexibly. The Transformer is an abstract component that is concretely implemented by the Triplifier and the RDFVirtualiser that will be introduced next. All possible implementations of Transformer include the IDataTransform interface, which is used by other components for accessing the transformation services provided by the component. The Transformer is able to retrieve metadata by interacting with the OpenDataCataloguer through the IMetaDataRetrieval interface. Similarly, it is enabled to access the data cleansing service through the IDataCleansing interface provided by the DataCleaner component, which is described further on.

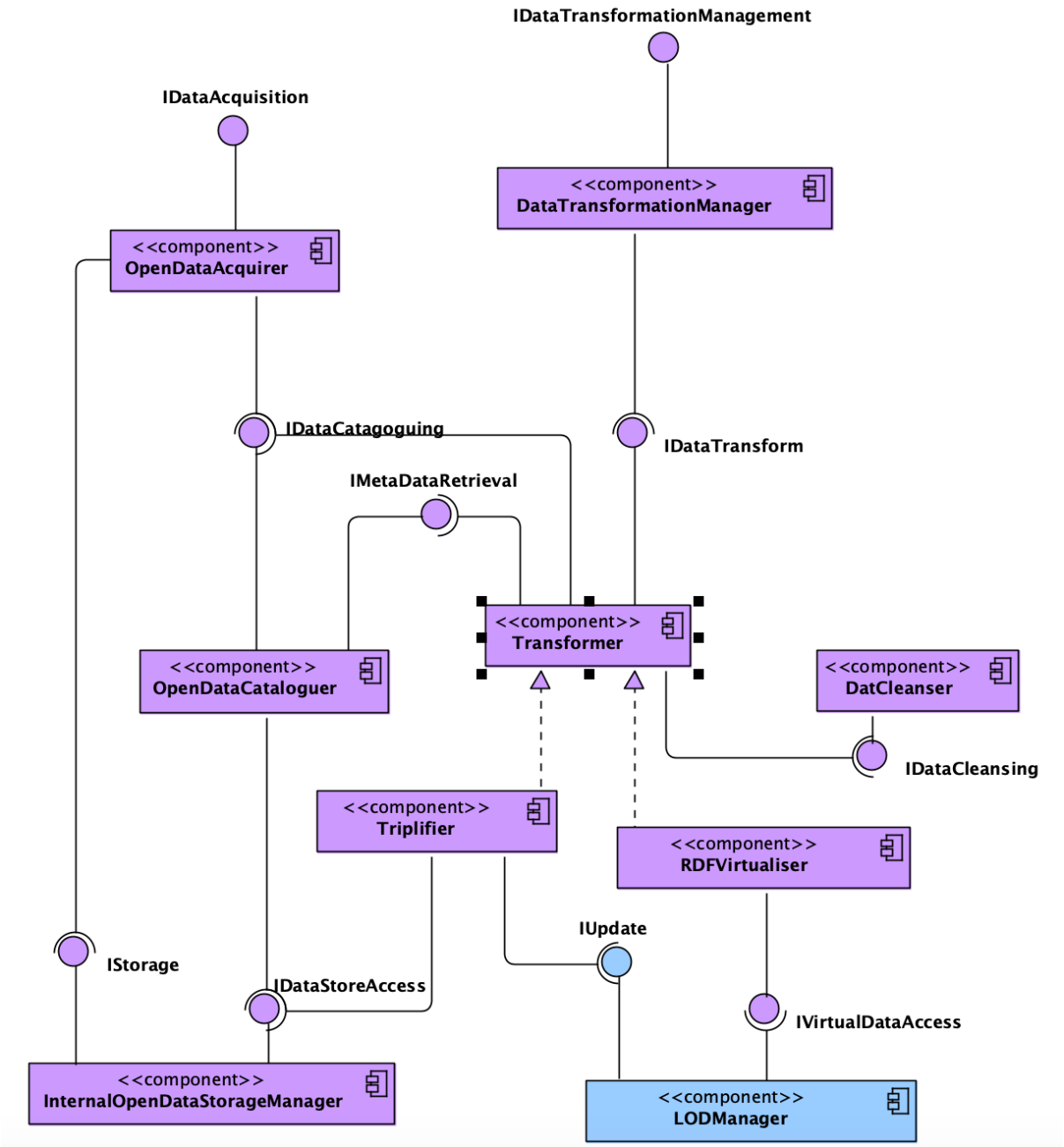

Figure 8: UML component diagram of the data preparation layer.

- **Triplifier:** the Triplifier is the implementation of the Transformer that performs a physical transformation of the original open data to RDF. Hence, after the transformation there are two versions of the same dataset, namely: (i) the original data and (ii) its RDF counterpart. The resulting RDF is then stored into a repository managed by the LODManager component through the IUpdate interface provided by the latter. It is worth noting that the LODManager component is part of the knowledge graph service management layer. Accordingly, such a

component is depicted in blue in Figure 8, which is the colour used for the aforementioned layer. In case the original data source is stored locally to WHOW it can be obtained by the Triplifier by asking the InternalOpenDataStorageManager to access such a data source via the interface IDataStoreAcess. On the contrary, if the original data source is maintained remotely it can be retrieved by means of the metadata that the Triplifier can access by means of the IMetaDataRetrieval interface that all implementations of Transformer require.

- **RDFVirtualiser:** in some situations in which having multiple versions of a same dataset is inconvenient, virtual RDF graphs might be more appropriate. In fact, virtual RDF graphs allow keeping the original datasource as-is without materialising RDF. This means that there is no need to store RDF and, accordingly, the maintenance is limited to the original dataset. The RDFVirtualiser is the component of the layer that enables WHOW to generate virtual read-only RDF graphs that can be used to query the original data source by means of SPARQL.
- **DataCleanser:** the DataCleanser provides services for cleansing data. Those services are focused on detecting and correcting/removing corrupt, inaccurate, incoherent facts from RDF. The DataCleanser provides access to the data cleansing services to other components through the IDataCleansing interface. In the layer the transformer is the only other component that interacts with the DataCleanser.
- **DataTransformationManager:** the DataTransformationManager is the component that is used for configuring, executing, and managing data transformations that involve a Transformer. The interaction with a Transformer is ensured by the interface IDataTransformer, which is provided by the latter and required by the DataTransformationManager. The DataTransformationManager can be accessed through its exposed interface labelled as IDataTransformationManagement. Such a component is also responsible for enabling the configuration of transformation strategies according to customisable criteria. For example, a dataset might be transformed incrementally or periodical updates might be set up.

5.4.2 Mapping with requirements and supporting technologies

The following table provides the requirements that are addressed by the components along with possible state of art technologies that might be exploited for implementing the components.

Component name	Addressed requirements	Possible technologies	Re-use strategy
OpenDataAcquirer	FR-03 (and all its sub	Dataverse[14], CKAN[15], Apache	To be designed and implemented by re-using

[14] https://dataverse.org/.

[15] https://ckan.org/.

	requirements), NFR-05	Cassandra[16], Apache ActiveMQ	existing technologies.
OpenDataCataloguer	FR-10, FR-14 (and all its sub requirements), FR-15 (and all its sub requirements)	CKAN[17], Socrata[18]	To be designed and implemented by re-using existing technologies.
InternalOpenDataStorageManager	FR-02, FR-04 (and all its sub requirements)	Dataverse[19], CKAN, Apache Cassandra, Socrata	To be designed and implemented by re-using existing technologies.
Transformer	FR-0401, FR-06, FR-07, NFR-04		To be designed and implemented by re-using existing technologies.
Triplifier	FR-0401, FR-06, FR-07, NFR-04	RML[20], RMLMapper[21], RMLStreamer[22], R2RML[23], pyRML[24], OpenLink Virtuoso (Sponger)[25] SDM-RDFizer[26] SPARQL anything[27]	To be designed and implemented by re-using existing technologies.

[16] https://cassandra.apache.org.

[17] https://ckan.org.

[18] https://dev.socrata.com/

[19] https://dataverse.org/.

[20] https://rml.io/specs/rml/.

[21] https://github.com/RMLio/rmlmapper-java

[22] https://github.com/RMLio/RMLStreamer

[23] https://www.w3.org/TR/r2rml/.

[24] https://github.com/anuzzolese/pyrml

[25] http://vos.openlinksw.com/owiki/wiki/VOS/VirtSponger

[26] https://github.com/SDM-TIB/SDM-RDFizer

[27] https://github.com/SPARQL-Anything/sparql.anything

		SPARQL Generate[28] ShExML[29]	
RDFVirtualiser	FR-0401, FR-06, FR-07, NFR-04	D2R[30], Stardog[31], OpenLink Virtuoso (Sponger)[32]	To be designed and implemented by re-using existing technologies.
DataCleanser	FR-01, FR-08	OpenRefine[33]	To be designed and implemented by re-using existing technologies.
DataTransformationManager	FR-06, FR-13, FR-12	Apache Stanbol[34], Apache ActiveMQ[35]	To be designed and implemented by re-using existing technologies.

Table 2: Mapping between requirements and supporting technologies for the Data Preparation layer's components

5.5 Knowledge graph layer

In order to address the specific requirement regarding the harmonisation of data content, including level of detail, where applicable (e.g. equivalent granularity in case of geospatial data), data structure, and semantics, the WHOW architecture offers a knowledge graph layer. Such a layer is not fully operational, that is, in contrast to the others, it does not mainly consist of software components but rather it employs a highly modular architecture of semantic resources that are represented by ontologies, reference data (controlled vocabularies) and the instances of them expressed in Linked Open Data, semantically linked to other LOD datasets available in the Web of Data. These resources,

[28] https://ci.mines-stetienne.fr/sparql-generate/

[29] https://github.com/herminiogg/ShExML

[30] http://d2rq.org.

[31] https://docs.stardog.com/.

[32] http://vos.openlinksw.com/owiki/wiki/VOS/VirtSponger

[33] https://openrefine.org/.

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

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

overall forming the knowledge graph of the data provider, are physically stored in a so-called triple store, the storage component of this layer. In the light of this observation, the section describes this layer from a methodological point of view, necessary to understand the required organisational activities that lead to the creation of robust semantic resources for data harmonisation purposes, and a UML component diagram that shows the abstraction defined for organising the various semantic resources (ontologies and data) into the triple store. The entire lifecycle of the resources is guaranteed by the software components of the other layers of the architecture.

As previously introduced, this layer is the most dependent on the specific application domain(s) of analysis.

The ontologies and controlled vocabularies of the layer are designed so as to be connected all together and organised in a network (see Ontology Network in Figure 7). The network provides a shared conceptualisation for harmonising the knowledge soup consisting of open datasets. These datasets are usually published with different syntaxes (e.g. CSV, XML, JSON, XLSX etc.) and modelled according to different semantics, which in most cases is implicit and hidden in the data itself. The ontologies and controlled vocabularies composing the knowledge graph layer enable a unified view that makes sense of such a knowledge soup, otherwise meaningless. Thus, it enables cross-domain and cross-border interoperability, otherwise impossible.

In this context, the design of the ontologies and controlled vocabularies for WHOW follows the eXtreme Design (XD) agile methodology (see also [2]) with both direct and indirect re-use of existing ontologies at national (e.g., Italian network of ontologies for Public Sector OntoPiA[36]) and European scale (e.g., EU Core Vocabularies[37]). For more details on the meaning of direct and indirect re-use of semantic assets in literature, the interested readers can refer to the scientific publication in [14].

As shown in Figure 9, which resumes in an UML diagram the XD methodology, the eXtreme design workflow is collaborative, as it requires the joint work of teams of ontology engineers and domain experts, this latter from data providers. The methodology is also incremental and iterative, as the design pairs integrate the modules produced by the other pairs to obtain incremental releases of the ontology.

[36] https://github.com/italia/daf-ontologie-vocabolari-controllati.

[37] https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/core-vocabularies

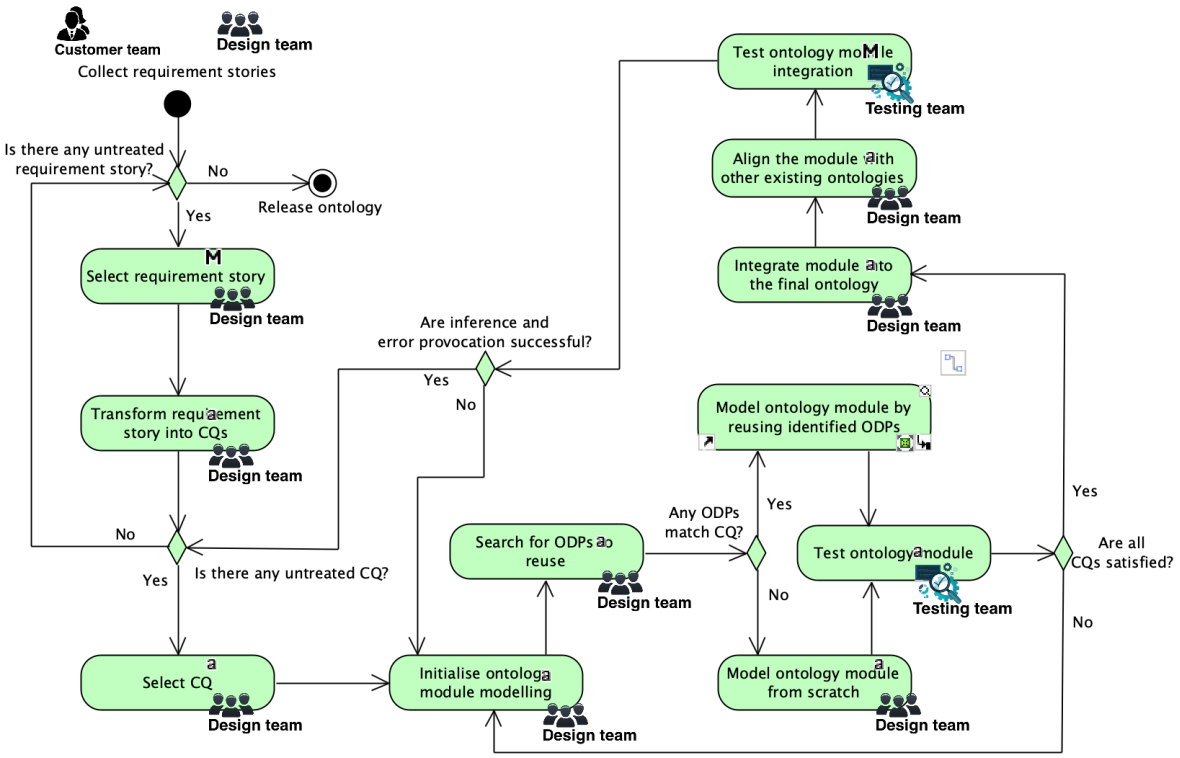

Figure 9: The eXtreme design iterative workflow.

The methodology is strongly based on the concept of Ontology Design Patterns (ODPs). ODPs are reusable modelling solutions created to solve recurrent ontological modelling issues[38]. We decided to use an ODP-based methodology as there are shreds of evidence [15] that the reuse of ODPs can (i) speed up the ontology design process, (ii) ease design choices, (iii) produce more effective results in terms of ontology quality, and (iv) boost (semantic) interoperability.

In the UML diagram of Figure 9, there are three teams that act in the design and creation of the ontologies/reference data of the WHOW knowledge graph: (i) the customer team that includes all the domain experts for water and health. The people in this team elicit the requirements that are afterwards mapped by the design team and testing team into ontological commitments ((ii) the design team consisting of ontology engineers who are not necessarily experts of the domain but they are able to represent its knowledge using knowledge engineering principles and best practices; (iii) testing team, possibly different from the people of the design team, responsible for all the testing activities.

[38] http://ontologydesignpatterns.org/wiki/Main_Page

The first action of the methodology is the collection of requirements elicited by the customer team together with the design team. At this step of the methodology, the requirements are recorded as stories, which are typically used in agile software development for governing the communication of requirements from the customer to the team responsible for developing the software.

After requirement stories are listed, the design team starts to transform them into so-called competency questions. With the term competency questions (CQs) we mean a natural language representation of the ontological commitments that drive the ontology development.

The next action in the methodology involves the design team that looks for possible ODPs to be used in order to respond to the CQs earlier identified. Those ODPs, if available, are reused for designing the modules of the ontology that address the specific CQs. When an ontology module is designed and implemented using the semantic web standards (e.g., OWL), the testing team performs the validation by assessing its fulfilment against the CQs. This validation is executed by (i) converting the CQs into SPARQL queries and (ii) executing those queries on a data sample that is represented according to the target ontology module.
If the validation is successful, the design team integrates the ontology module in the final ontology. Additionally, the design team provides alignments with related external ontologies and vocabularies in the Semantic Web for (semantic) interoperability purposes. Then, the testing team performs a set of integration tests that aim at (i) validating the logical consistency of the ontology and (ii) its ability to detect errors by deliberately introducing contradictory facts [16].
If the integration tests succeed, then the design team performs another iteration of the process by selecting an untreated CQ. If no untreated CQ is available, then the iteration consists of the selection of an untreated requirement story. Finally, after several iterations and when no other untreated requirement story is available, the process ends and the stable version of the ontology is released.
After a preliminary analysis we have already conducted during co-creation programme meetings on some WHOW datasets, identified in order to support the business use cases defined in deliverable D2.1, we were able to select a set of existing ODPs that can be highly relevant for our modelling purposes. These ODPs are:

1. Observation - measure design pattern, which is also proposed as an INSPIRE data model of reference[39];
2. Classification design pattern[40];
3. Spatial object: features and geometry design pattern, which is proposed by the standard GeoSPARQL of OGC (Open Geospatial Consortium)[41];
4. Pollution design pattern[42].

5.5.1 Layer's software components

[39] https://inspire.ec.europa.eu/documents/Data_Specifications/D2.9_O&M_Guidelines_v2.0rc3.pdf

[40] http://ontologydesignpatterns.org/wiki/Submissions:Classification

[41] https://www.ogc.org/standards/geosparql

[42] http://ontologydesignpatterns.org/wiki/Submissions:Pollution.

The component diagram of this layer is illustrated in the following Figure 10.

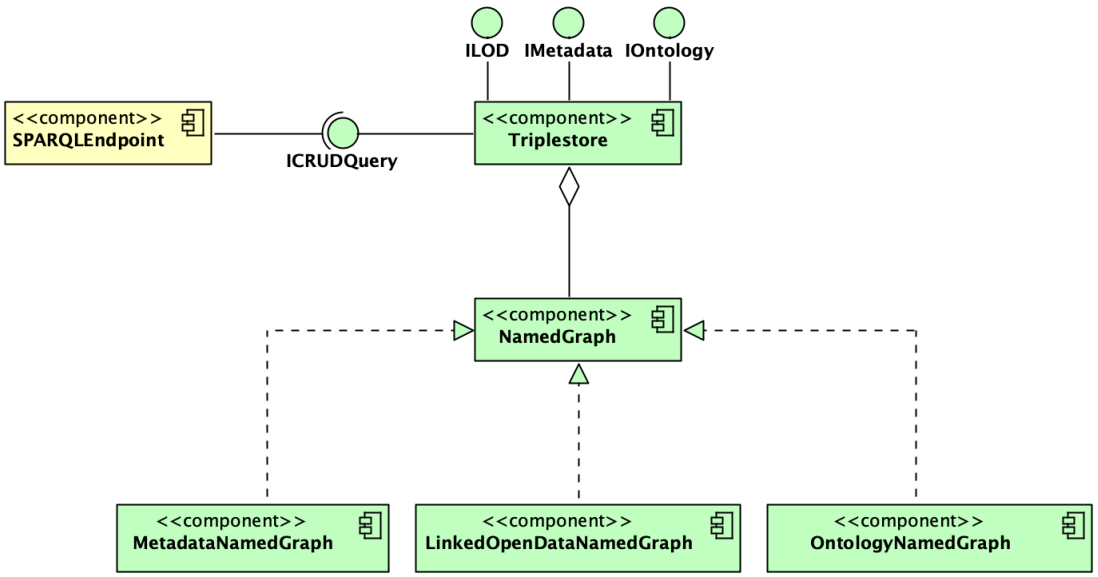

Figure 10: UML component diagram of the knowledge graph layer

The main component is represented by the storage that allows data providers to store any data, including ontologies in the form of triples `subject-predicate-object`. This component is named **TripleStore** and provides all CRUD (create, read, update and delete) functions (ICRUDQuery Interface in Figure 10) at the level of single entities, single triples, sets of entities, sets of triples, but also single graphs (i.e. named graph) and sets of graphs. In the design of the WHOW architecture we plan to organise the content of the triple store in so-called named graphs. A named graph is a set of RDF statements that is given a URI to uniquely identify it.. Named graphs allow us to separate, within the triple store, the data from the ontology network resources and metadata. In Figure 10, we represent that a TripleStore aggregates a component named **NamedGraph** which is realised through three components; namely, **MetadataNamedGraph**, which is the set of RDF statements related to the descriptive metadata being defined according to (Geo)DCAT-AP, **LinkedOpenDataNamedGraph**, which is the set of RDF statements involving of the Linked Open Data of the knowledge graph, including reference data or controlled vocabularies, and the **OntologyNamedGraph**, which is the named graph for the ontology part of the knowledge graph, only.

According to this organisation of the triple store, this latter component can then expose three types of interfaces, ILOD, IMetadata, IOntology so as to control the access to the various named graphs by other components of the architecture.

The triple store directly communicates with the SPARQLEndpoint component of the application layer through the ICrudQuery interface offered by the triple store.

5.5.2 Mapping with requirements and supporting technologies

The following table provides the requirements that are addressed by the components along with possible state of art technologies that might be exploited for implementing the components.

Component name	Addressed requirements	Possible technologies	Re-use strategy
Triplestore	FR-0401, FR-05, FR-44, FR-45, FR-46	Virtuoso RDF[43], GraphDB[44] , Stardog[45], neo4j[46]	Off-the-shelf
NamedGraph	FR-05, FR-06, FR-16, FR-17	Virtuoso RDF, GraphDB , Stardog, neo4j	To be properly configured in the Triple store
MetadataNamedGraph	FR-05, FR-06, FR-16, FR-17	Virtuoso RDF, GraphDB , Stardog, neo4j	To be properly configured in the Triple store
LinkedOpenDataNamedGraph	FR-05, FR-06, FR-16, FR-17	Virtuoso RDF, GraphDB , Stardog, neo4j	To be properly configured in the Triple store
OntologyNamedGraph	FR-05, FR-06, FR-16, FR-17	Virtuoso RDF, GraphDB , Stardog, neo4j	To be properly configured in the Triple store

Table 3: Mapping between requirements and supporting technologies for the Knowledge graph layer's components

[43] https://virtuoso.openlinksw.com

[44] https://graphdb.ontotext.com

[45] https://www.stardog.com/

[46] https://neo4j.com/

In addition to these technologies, it is worth mentioning the possible tools that are usually adopted for ontology and reference data creation purposes. In particular, Protégé[47] is a well-known open source editor for ontologies design. It includes a variety of plugins for ontology design patterns annotation and usage, reasoning and validation, also through standard languages such as SHACL.

Reference data, or controlled vocabularies, are usually represented in the Linked (Open) Data paradigm using SKOS Web standard and tools such as VocBench[48] can be effectively used to support the creation of this type of semantic resource of the WHOW knowledge graph.

5.6 Knowledge graph service management layer

This layer implements a set of components that wrap the knowledge graph by providing knowledge-intensive services that interact with the ontology network, the metadata and the Linked Open Data. This wrapping aims at abstracting the way in which knowledge is physically represented and maintained into the knowledge graph layer from the way knowledge can be accessed and used in business-oriented services.

5.6.1 Layer's software components

In Figure 11 we model the knowledge graph service management layer as composed of the following components and their corresponding interfaces:

- **LODManager:** the LODManager is the component designed on top of the triplestore in order to abstract CRUD [5] operations performed on the storage physically. This allows the LODManager to act as the mediator of transparent interactions between any component/application (i.e. knowledge client) that needs to access the WHOW Linked Open Data (LOD) and the triplestore (i.e. knowledge repository). Accordingly, the knowledge client does not need to know any specification about the underlying repository. This addresses the design of low coupled components, meaning that they are as independent as possible from each other, so that changes to one of them do not heavily impact other components. More specifically, on one hand the LODManager accesses the CRUD operations over the LOD exposed by the Triplestore by requesting the ILOD interface provided by the latter. On the other hand, the LODManager exposes querying capabilities over the Triplestore to other components by providing them with (i) the IQuery interface for SELECT queries and (ii) the

[47] https://protege.stanford.edu/.

[48] http://vocbench.uniroma2.it/.

IUpdate interface for UPDATE queries. We remark that the standard languages that are adopted for SELECT and UPDATE queries are SPARQL 1.1[49] and SPARQL 1.1. Update[50].

- **MetadataManager:** the MetadataManager is designed with the similar intent as the LODManager of providing a mediator for abstracting the interaction with the Triplestore limited to the management of metadata. Hence, it is the component that enables other components, such as the OpenDataCataloguer, to create, modify, and delete metadata transparently with respect to the actual implementation of the physical storage, which might depend on specific needs that can be known only at deployment time. The MetadataManager provides the IMetadataManagement interface to other components and requests the IMetadata interface to the Triplestore.

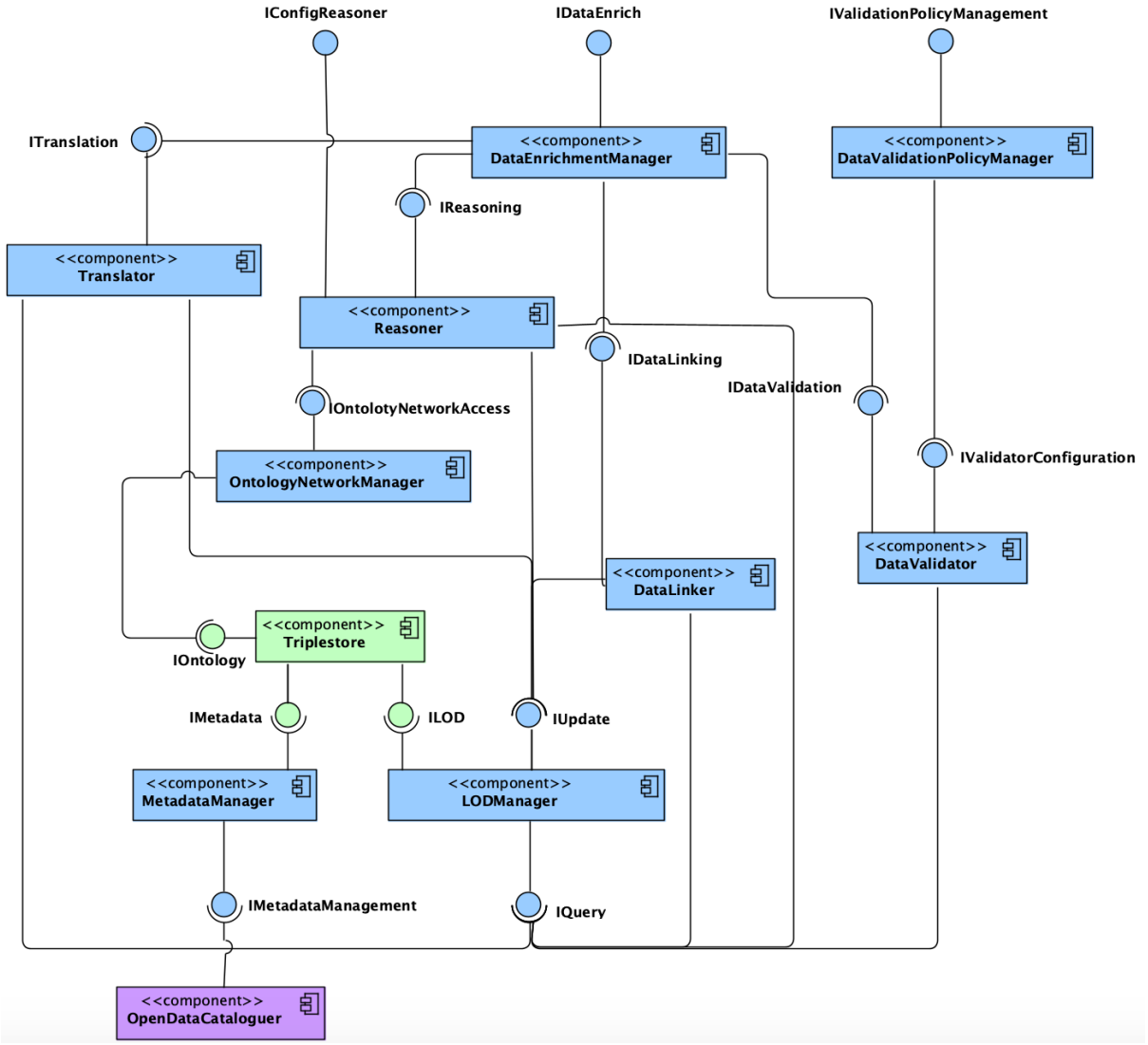

Figure 11: UML component diagram for the Knowledge Graph Service Management Layer

[49] https://www.w3.org/TR/sparql11-query/

[50] https://www.w3.org/TR/sparql11-update/

- **OntologyNetworkManager:** the OntologyNetworkManager is in charge of providing the management functionality on top of the ontology network. The ontology network is stored in the triplestore. Hence, OntologyNetworkManager abstracts the ontology network into a software component. It interacts with the Triplestore by requesting the IOntology interface provided by the latter. On the contrary, it interacts with any other client component by providing them with the IOntologyNetworkAccess interface.
- **Translator:** the translator is the component of the layer that implements the translation services for enriching the knowledge graph with multilingual literals. It is of utmost importance that the eTranslation building block of Digital Service Infrastructure is integrated in this component to provide data, conceptualisations and any other information in official European languages. The Translator accesses the data by querying the knowledge graph through the IQuery interface provided by the LODManager. The knowledge graph is accessed in order to retrieve knowledge blocks (e.g. specific sets of literals associated with relevant predicates such as rdfs:comment, rdf:label, etc.), which are meaningful and worthy to be translated for boosting the multilinguality of the knowledge graph. Once the data are translated then the Translator can update the knowledge graph with the output of the translation. This can be done by requesting the IUpdate interface provided again by the LODManager. The Translator provides the ITranslation interface, which is requested by the DataEnrichmentManager. The latter is the component aimed at providing a unified access point to the components of the upper layer that want to transparently use the enrichment functionalities exposed by the different components belonging to this layer (e.g. translation, reasoning, validation, etc.).
- **Reasoner:** the reasoner provides the framework with reasoning capabilities. These include description logics [9] (DL) reasoning and any pertinent inductive, deductive, or abductive reasoning based on the axioms defined in the ontology network. The reasoner can be configured through the interface IConfigReasoner. The configuration allows client components to set specific reasoning properties, e.g. the exploitation of the transitive closure of subclasses (i.e. rdfs:subClassOf axioms) for enabling hierarchy inference, etc. Instead, the Reasoner provides the IReasoning interface for allowing other components to perform reasoning tasks.
- **DataLinker:** creating qualitative Linked Open Data (i.e. the five stars of Linked Data[51]) requires, besides the other commitments (e.g. use of HTTP(s), RDF, and SPARQL), to create links towards other LOD. The DataLinker is the component meant to cope with this need. In fact, it provides capabilities to the WHOW framework for the discovery of links between the LOD in WHOW and those available in the Web of Data. Those links can be discovered by applying a variety of techniques and tools, such as, string similarity metrics [6] (e.g. cosine similarity), machine learning [7] (e.g. bayesian methods), or deep learning [8] (recurrent neural networks). On one hand, the linking capabilities of the DataLinker can be accessed through the IDataLinking interface. On the other hand, the DataLinker requires the IQuery

[51] https://www.w3.org/2011/gld/wiki/5_Star_Linked_Data

and IUpdate interfaces provided by the LODManager for retrieving data to be linked to external ones and update the knowledge graph with new links, respectively.

- **DataValidator:** The DataValidator is responsible to perform data validation, which is core in order to publish coherent, consistent, and sound LOD. The validation is executed by applying predefined and customisable validation policies. The DataValidator provides two interfaces, namely: (i) the IDataValidation that can be required by any component that needs to validate data and (ii) IDataValidationConfiguration that allows the DataValidator to be configured in terms of validation policies. The data to be validated are requested to the LODManager through its IQuery interface.
- **DataValidationPolicyManager:** The validation policies that are applied by the DataValidator are created and managed by the DataValidationPolicyManager. Those policies can be represented by using the Shape Constraint Language[52] (SHACL). The DataValidationPolicyManager can be accessed through the IValidationPolicyManagement interface and interacts with the DataValidator by accessing its IValidatorConfiguration.
- **DataEnrichmentManager:** this component aims at aggregating all the enrichment services available in the layer for exposing them in a unified way to the upper layer in order to allow the latter to access the services homogeneously. The enrichment services are all those services that enrich the LOD with additional data not previously part of the original LOD. Namely, those services are provided by: the (i) DataLinker, (ii) DataValidator, (iii) Translator, and (iv) Reasoner. We refer to these components as the enrichment components. The DataEnrichmentManager interacts with the aforementioned components through the (i) IDataLinking, (ii) IDataValidation, (iii) ITranslation, and (iv) IReasoning interfaces, respectively. The homogeneous access to the enrichment services is enabled to the upper layer by means of the IDataEnrich interface, which, in turn, is provided by DataEnrichmentManager. Hence, the DataEnrichmentManager is responsible for mediating the interaction between the components of the upper layer with the components of this layer that provide the enrichment services. This mediation is also performed by managing the tasks executed by the enrichment components as an asynchronous job. This allows the WHOW framework to cope with long-lasting enrichment task executions (e.g. DL reasoning) that might occur in case of big data. This means that a component that requests the execution of an enrichment task to the DataEnrichmentManager through the IDataEnrich interface is provided with an enrichment job identifier. The latter can be used for requesting the status of the enrichment job. Anyway, the component that requested the execution of an enrichment job is always notified when such a job ends. Finally, the job identifier can be used for retrieving the output of the enrichment asynchronously.

5.6.2 Mapping with requirements and supporting technologies

The following table provides the requirements that are addressed by the components along with (i) possible state of art technologies that might be exploited for implementing the components and (ii) a re-use strategy to rely on for the implementation.

[52] https://www.w3.org/TR/shacl/

Component name	Requirement	Possible technology	Re-use strategy
LODManager	FR-02, FR-08, FR-21	LDP[53]	To be designed and implemented by re-using existing technologies.
MetadataManager	FR-10, FR-15 (and all its sub requirements)	CKAN	To be designed and implemented by re-using existing technologies.
OntologyNetworkManager	FR-0401, FR-11 (and all its sub requirements), NFR-04	Apache Stanbol	To be designed and implemented by re-using existing technologies.
Translator	FR-34	eTranslation CEF building block[54] OpenNMT[55] LibreTranslate[56]	Preferably off-the-shelf. In case no valid ready-to-use open-source solution is found then the component is to be designed and implemented by re-using existing technologies such as OpenNMT.
Reasoner	FR-0401, FR-06,	Apache Stanbol,	Off-the-shelf

[53] https://www.w3.org/wiki/LDP_Implementations#LDP.js_.28Server.29

[54] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eTranslation

[55] https://opennmt.net/

[56] https://github.com/LibreTranslate/LibreTranslate

	NFR-04	HermiT[57] [12], Pellet[58] [13]	
DataLinker	FR-09, FR-15, FR-17	Silk[59] [10], LIMES[60] [11]	Off-the-shelf
DataValidator	FR-11 (and all its sub requirements)	rdf4j[61], Apache Jena[62], DCAT-AP Validator[63]	Off-the-shelf (it can also be designed and implemented starting from validators available at the European level in the context of application profiles of EU core vocabularies)
DataValidationPolicyManager	FR-11 (and all its sub requirements)	SHACL[64]	Off-the-shelf
DataEnrichmentManager	FR-01	Apache Stanbol	Off-the-shelf

Table 4: Mapping between requirements and supporting technologies for the Knowledge graph service management layer's components

5.7 Application layer

[57] http://www.hermit-reasoner.com/

[58] https://github.com/stardog-union/pellet

[59] http://silkframework.org/

[60] https://aksw.org/Projects/LIMES.html

[61] https://rdf4j.org/

[62] https://jena.apache.org/documentation/shacl/index.html

[63] https://www.itb.ec.europa.eu/shacl/dcat-ap/upload

[64] https://www.w3.org/TR/shacl/

The application layer is the foundational layer for all use cases related to data consumption by any stakeholder interested in exploring the WHOW knowledge graph. In this layer in fact we define the software components of WHOW architecture that are responsible for application functionalities to be delivered to end-users mainly through knowledge graph access and querying. The UML diagram of Figure 12 illustrates all the software components of this layer and their interactions through specific identified interfaces. The diagram also highlights relevant interactions of this layer with components of the presentation (Section 4.6) and knowledge graph layers (Section 4.4).

5.7.1 Layer's software components

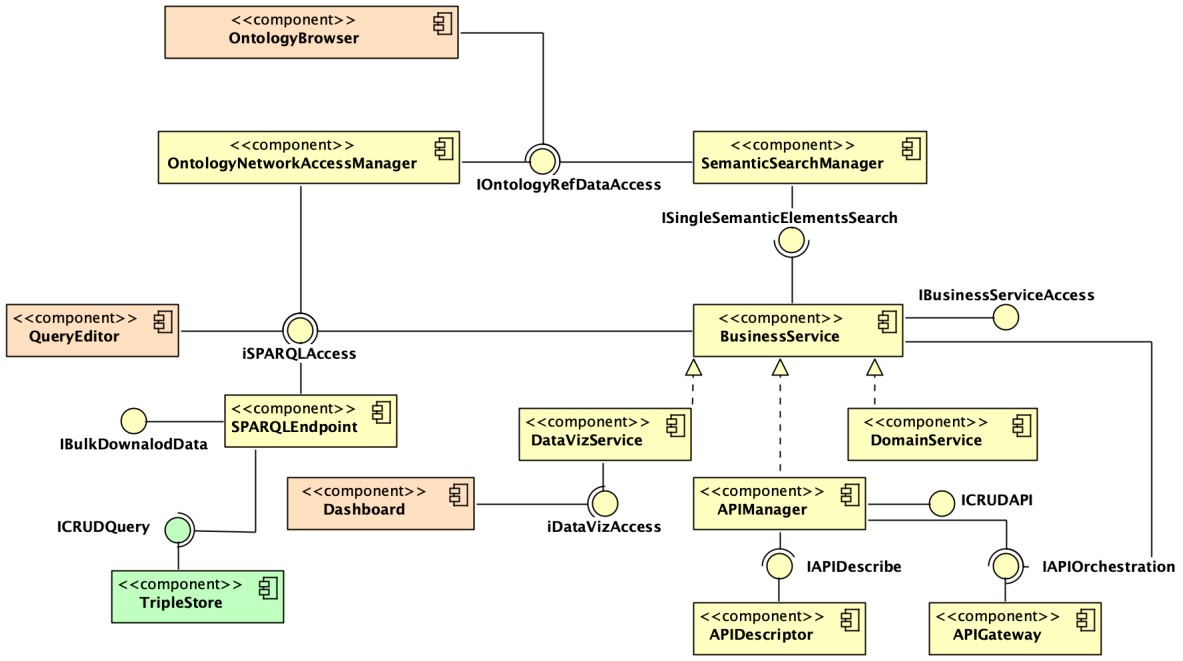

Figure 12: UML Component diagram of the application layer

In Figure 12 we model the application layer as formed by the following components and their corresponding interfaces:

- **SPARQLEndpoint**: a SPARQL endpoint is a point of access that implements the SPARQL query language and protocol, used in the Linked Open Data paradigm to query RDF triples stored in so-called triple stores. This component is the core element of this layer of the WHOW architecture. It is through this that data access and querying operations are possible for all the other software components.

It exposes two interfaces: iSPARQLAccess and iBulkDownloadData. The first interface is used in all the cases in which a specific query defined in SPARQL on the knowledge graph, meaning ontology network, reference data and linked open data as shown in Figure 7, is to be carried out. This interface is usually invoked by the presentation layer of the SPARQL endpoint that is represented in Figure 12 as the QueryEditor component, a specific view of the User Interface (see section 4.6). The QueryEditor component in fact allows users to define and run SPARQL queries that are then redirected to the SPARQLEndpoint via the iSPARQLAccess interface.

The second interface enables the possibility for downloading parts of the knowledge graph in bulk, an important functionality of the open data paradigm. This interface is particularly useful to support all those use cases where an end-user browses the list of bulk datasets made available by a data provider in the form of a catalogue, and then downloads them for further elaborations, local to his/her machine for example.

The SPARQLEndpoint operates in strict collaboration with the actual triple store of the knowledge graph layer; it invokes CRUD operations (iCRUDQuery Interface in Figure 12 offered by the knowledge graph layer's Triplestore component) on the store according to the types of request of access done by other components.

- **OntologyNetworkAccessManager**: it is the component that enables the access to the ontology network for the presentation layer and for all the other components that need to support specific searches on the semantic data models created by WHOW. Since the ontology network is a piece of the knowledge graph (OntologyNamedGraph of Figure 10), and it is stored in the triple store together with the linked open data, this component uses the ISPARQLAccess of the SPARQLEndpoint in Figure 12 to get access to the related named graph via SPARQL. The OntologyNetworkAccessManager provides itself an interface we named IOntologyRefDataAccess that is an interface used by other applications part of this layer that needs to get access to the ontological elements, including any reference data or controlled vocabulary, of WHOW knowledge graph. For example, the OntolgyBrowser of the presentation layer, that is a specific view of the user interface, can use the interface IOntologyRefDataAccess to present ontological elements to end-users;
- **SemanticSearchManager**: it is a component particularly relevant for enabling searches of semantic assets and of any other element in the knowledge graph that is expressed with a clear semantics. It can be viewed as an enabler for the construction of advanced services that might want to leverage semantic search functionalities. It uses the IOntologyRefDataAccess interface of the previous component in order to get access to the ontology part of WHOW knowledge graph, and it makes available, through the interface ISingleSemanticElementSearch, the possibility to search on single semantic elements of the knowledge graph.

  Please note that, this service is not a mandatory service; in essence, its availability is an added value and can serve different usage scenarios also for presentation purposes; however, it does not directly impact on the availability of the knowledge graph at data provider that in turn contributes to the construction of the overall WHOW open knowledge graph;

- **BusinessService:** this component represents any type of service that is more related to the business aspect of WHOW open data. In contrast to other components of this and other

layers, it is strongly dependent on the application domains under evaluation, that is, water and health. In Figure 12, this component is an abstract service that may include:

- a **DataVizService,** which is responsible for providing data visualization and stories of WHOW linked open data. Please note that this component is not a mandatory for achieving the objectives of the WHOW project; however, it can be a valuable support for narrating the data offered by the various WHOW data providers;
- a **DomainService,** which is a very generic component that can be instantiated in a variety of ways according to the specific application domain service to offer to the end-users. As in the previous case, this component is not mandatory for meeting the overall WHOW objectives; it has been introduced in the Linked Open Data Reference Architecture we propose in this deliverable as an example of the value of providing a knowledge graph, which is fully open for the maximum re-use, thus enabling the development of any type of DomainService.
- a **APIManager**, which is, in contrast to the previous component, a crucial part of the WHOW architecture and thus a mandatory component. Through this architectural element, WHOW effectively enables the machine-to-machine interactions with external systems possibly owned by re-users, also not fully familiar with semantic web standards but used to develop applications and/or services using the Rest paradigm. The availability of such a software element allows WHOW data providers to foster a wider use of the data by anyone.

  This component is responsible for the creation and maintenance of the set of Application Programming Interfaces (APIs), according to the REST paradigm, derived from the WHOW knowledge graph. It interacts with two other important components of this layer; namely, the APIDescriptor and APIGateway. The **APIDescriptor** is used to represent documentation of the offered API to end-users. It exposes the IAPIDescribe interface used by other components and in the user interface to understand the API and how to invoke it. The **APIGateway** is a flexible abstraction layer and a single entry point that securely manages the communications between client software components and all the different API URLs offered by the WHOW architecture according to the main concepts of the WHOW knowledge graph.

5.7.2 Mapping with requirements and supporting technologies

The following table provides the mapping with the requirements that are addressed by the components along with possible state of art technologies that might be exploited for implementing the components. The re-use strategy for these technologies is also introduced.

Component name	Requirement	Possible technology	Re-use strategy
SPARQLEndpoint	FR-37 (and all its sub requirements), FR-38, FR-39, FR-41, FR-42, FR-43	Virtuoso, GraphDB, StarDog, Amazon Neptune (possibly in their open source versions)	Off-the-shelf. We will leverage existing endpoints of data providers, if available, and will strengthen their capabilities to support a wide range of queries, if required.
OntologyNetworkAccessManager	FR-12, FR-25, FR-26, FR-27, FR-28, 29, FR-30, FR-31, FR-32, FR-33, FR-34, FR-35, 36	LodView/LodLive, Lode	To properly configure existing open source tools
SemanticSearchManager	FR-26, FR-27, FR-29	It can be constructed based on the use of open software such as Apache Stanbol, Solr, ElasticSearch	To be designed and implemented by relying on state of the art open-source solutions, e.g. Solr.
DataVizService	FR-18 (and all its sub requirements), FR-19, FR-20, FR-24	D3.js[65], PowerBI,[66] Graphana[67], Superset[68], Metabase[69]	Off-the-shelf

[65] https://d3js.org/

[66] https://powerbi.microsoft.com/en-au/

[67] https://grafana.com/

[68] https://superset.apache.org/

[69] https://www.metabase.com/

DomainService	N.A	-	To be designed and implemented by relying on possibly state of the art open-source tools
APIManager	FR-37, FR-38, FR-39, FR-43	Lizard[70], OBA[71], R4R[72], LDP[73]	It can be possible to re-use existing components with proper configurations according to WHOW knowledge graph content
APIDescriptor	FR-37, FR-38, FR-39, FR-43	Swagger[74], LDP	It can be possible to re-use existing components with proper configurations according to WHOW knowledge graph content
APIGateway	FR-37, FR-38, FR-39, FR-43	KongHQ[75]	To be designed and implemented by relying on state of the art open-source solutions

Table 4: Mapping between requirements and supporting technologies for Application layer's components

[70] https://github.com/anuzzolese/lizard.

[71] https://oba.readthedocs.io/en/latest/.

[72] https://github.com/TBFY/r4r.

[73] https://www.w3.org/TR/ldp/

[74] https://swagger.io/

[75] https://konghq.com/

5.8 Presentation layer

The presentation layer of the WHOW architecture embodies the actual user interfaces for human and machine-to-machine interactions. As depicted in Figure 7, it is the layer that directly enables the communications with software agents, through APIs, and human-beings through Web interfaces.

5.8.1 Layer's software components

Figure 13 illustrates the UML component diagram of this layer.

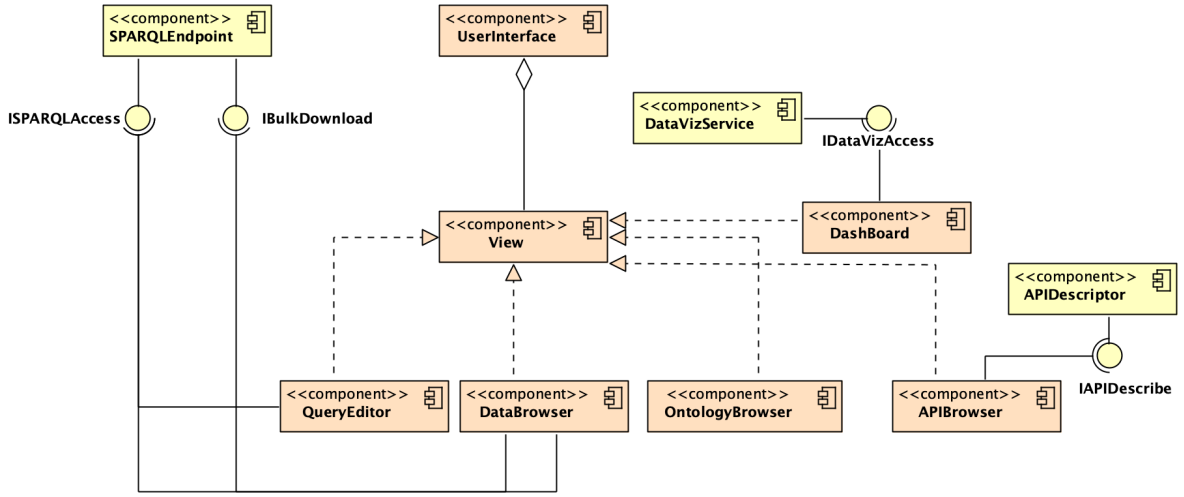

Figure 13: UML component diagram of the presentation layer

As shown in Figure 13, the main component of this layer is represented by the **UserInterface** that aggregates the component named **View,** which is in turn realized through five main components:

- **QueryEditor**, which is usually offered by SPARQL endpoints to enable human-interaction services for querying the Linked (Open) Data through the SPARQL protocol. The QueryEditor in fact uses the ISPARQLAccess interface of the SPARQLEndpoint component;
- **DataBrowser,** which is the Web-based view that allows users to browse both datasets, downloadable in bulk (it uses in fact the IBulkDownload interface provided by the SPARQLEndpoint component) and punctual instances of Linked Open Data of the WHOW knowledge graph, through the use of the ISPARQLAccess interface;
- **OntologyBrowser,** the Web-based view that enables users to browse the ontologies and controlled vocabularies of the WHOW knowledge graph;
- **APIBrowser**, which is the Web-based view that allows users to browse the APIs offered by WHOW and described through the APIDescriptor component of the application layer;

- **Dashboard**, which is the Web-based view of the DataVizService component that allows users to have at-a-glance insights of key performance indicators (KPIs) relevant for the WHOW application domains.

5.8.2 Mapping with requirements and supporting technologies

The following table provides the requirements that are addressed by the components along with possible state of art technologies that might be exploited for implementing the components.

Component name	Requirement	Possible technology	Re-use strategy
QueryEditor	FR-37, 39, 40, 41, 42	Triply[76], Query editors offered by such tools as Virtuoso, GraphDB, etc.	Off-the-shelf
DataBrowser	FR-16, FR-18, FR-19, FR-20, FR-21, FR-22, FR-23, FR-24, FR-35, FR-36	Web interface offered by tools such as LodView[77]	To be designed and implemented by relying on state of the art open-source solutions
OntologyBrowser	FR-25, FR-26, FR-27, FR-28, FR-29, FR-30, FR-31, FR-32, FR-33, FR-35, FR-36	Web interface offered by tools such as LodView, LODE	To be designed and implemented by relying on state of the art open-source solutions
APIBrowser	FR-36, FR-37,	Swagger Editor[78]	To be designed and

[76] https://yasgui.triply.cc/.

[77] https://lodview.it/.

[78] https://editor.swagger.io/

	FR-43		implemented by possibly relying on state of the art open-source solutions
Dashboard	FR-18, FR-19, FR-20, FR-25	Types of graphics offered by such tools as Graphana, Superset, Metabase, etc.	Off-the-shelf and also ad hoc implementation is possible

Table 5: Mapping between requirements and supporting technologies for Presentation layer's components

6. Conclusions

This deliverable introduced the design of the Linked Open Data reference architecture to be deployed and used at each data provider site.

The architecture allows data providers to produce, manage and publish knowledge graphs on water consumption and pollution and health parameters. Thanks to native semantic links among data to be enabled, the knowledge graphs can be federated, forming the so-called WHOW knowledge graph.

According to software development best practices, an analysis of the different technical use cases has been carried out and described in the deliverable. Each use case is presented using the UML modelling language, clearly identifying the actors involved in it and all the conditions that contribute to describe it. Functional and non-functional requirements are then listed and used for the design of the WHOW Linked Open Data reference architecture that the deliverable introduces.

The architecture is logically organised in five layers, each of which embodies a set of software and resource components. All the resources and software are made available as open artefacts with open licenses, thus facilitating the re-use by anyone.

References

1. European Data Portal. *What is open data*. Retrieved 11/10/2021. https://data.europa.eu/en/trening/what-open-data.
2. Valentina Presutti, Enrico Daga, Aldo Gangemi, and Eva Blomqvist. “eXtreme Design with content ontology design patterns”. In Eva Blomqvist, Kurt Sandkuhl, Francois Scharffe, and Vojtech Svatek, editors, *Proceedings of the Workshop on Ontology Patterns* (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009, volume 516. CEUR Workshop Proceedings, 2009.
3. Tim Berners-Lee. *Linked data. Design issues.* W3C. 27/07/2006. Retrieved 11/10/2021. https://www.w3.org/DesignIssues/LinkedData.html
4. Solid - Social Linked Data - https://solidproject.org/.
5. James Martin. Managing the data base environment. Prentice Hall PTR, 1983.
6. Manel Achichi, Mohamed Ben Ellefi Zohra Bellahsene, and Konstantin Todorov. "Linking and disambiguating entities across heterogeneous RDF graphs." Journal of Web Semantics 55 (2019): 108-121.
7. Andriy Nikolov, Mathieu d’Aquin, and Enrico Motta. "Unsupervised learning of link discovery configuration." In Extended Semantic Web Conference, pp. 119-133. Springer, Berlin, Heidelberg, 2012.
8. Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. "Deep learning for entity matching: A design space exploration." In Proceedings of the 2018 International Conference on Management of Data, pp. 19-34. 2018.
9. Franz Baader, Ian Horrocks, and Ulrike Sattler. "Description logics." Foundations of Artificial Intelligence 3 (2008): 135-179.
10. Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. "Silk-a link discovery framework for the web of data." In Ldow. 2009.
11. Axel-Cyrille Ngonga Ngomo, and Sören Auer. "LIMES—a time-efficient approach for large-scale link discovery on the web of data." In 22nd International Joint Conference on Artificial Intelligence. 2011.
12. Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. "HermiT: an OWL 2 reasoner." Journal of Automated Reasoning 53, no. 3 (2014): 245-269.
13. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. "Pellet: A practical owl-dl reasoner." Journal of Web Semantics 5, no. 2 (2007): 51-53.

14. Valentina Presutti, Giorgia Lodi, Andrea Nuzzolese, Aldo Gangemi, Silvio Peroni, Luigi Asprino. “The role of ontology design patterns in linked data projects”, in Proceedings of the International Conference on Conceptual Modeling, 2016.
15. E. Blomqvist, V. Presutti, E. Daga, A. Gangemi, Experimenting with eXtreme design, in Proceedings of the 17th International Conference on Knowledge Engineering and Knowledge Management, ed. by P. Cimiano, H.S. Pinto Lecture Notes in Computer Science, vol. 6317, Springer, 2010, pp. 120–134. Springer. doi:10.1007/978-3-642-16438-5 9
16. E. Blomqvist, A. Seil Sepour, V. Presutti, Ontology Testing - Methodology and Tool, in Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management, ed. by A. ten Teije, J. Volker, S. Handschuh, H. Stuckenschmidt, M. dAcquin, A. Nikolov, N. Aussenac-Gilles, N. Hernandez Lec
17. D. Garlan, R.T. Monroe and D. Wile, Acme: Architectural description of component-based systems, in: Foundations of Component-Based Systems, G.T. Leavens and M. Sitaraman, eds, Cambridge University Press, 2000, pp. 47–68.
18. R. T. Fielding. REST: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.