Semantic Web 1 (0) 1–5
IOS Press

Helio: a framework for implementing the life cycle of knowledge graphs

Andrea Cimmino [a] and Raúl García-Castro [a]
[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, ES, Spain*
E-mail: {andreajesus.cimmino, r.garcia}@upm.es

Editor: Aidan Hogan, University of Chile, Chile
Solicited reviews: Umutcan Simsek, University of Innsbruck, Austria; Two anonymous reviewers

Abstract. Building and publishing knowledge graphs (KG) as Linked Data, either on the Web or in private companies, has become a relevant and crucial process in many domains. This process requires that users perform a wide number of tasks conforming to the life cycle of a KG, and these tasks usually involve different unrelated research topics, such as RDF materialisation or link discovery. There is already a large corpus of tools and methods designed to perform these tasks; however, the lack of one tool that gathers them all leads practitioners to develop ad-hoc pipelines that are not generic and, thus, non-reusable. As a result, building and publishing a KG is becoming a complex and resource-consuming process. In this paper, a generic framework called Helio is presented. The framework aims to cover a set of requirements elicited from the KG life cycle and provide a tool capable of performing the different tasks required to build and publish KGs. As a result, Helio aims at providing users with the means for reducing the effort required to perform this process and, also, Helio aims to prevent the development of ad-hoc pipelines. Furthermore, the Helio framework has been applied in many different contexts, from European projects to research work.

Keywords: Knowledge graph generation, Knowledge graph publication, Linked data

1. Introduction

The presence of knowledge graphs (KGs) published openly on the Web, or privately as Linked Data has grown in the last decade [1]. The reason for this growth is due to the fact that many domains demand data to be published homogeneously under a common representation which, sometimes, requires translating existing heterogeneous data from a set of data sources [2]. To this end, the data of the KGs could be built using Semantic Web Technologies [3] like RDF and, then, published following the Linked Data principles [4]. Building a KG is not a simple process since it may involve many tasks that belong to different research topics [5]; from the translation of data into RDF using materialisers [6], to the generation of links among the resources of different KGs utilizing link discovery tools [7].

Numerous tools aim at performing one or more tasks, which are related to these research topics, for building and publishing a KG [8]. However, these tools were designed with a narrow scope that aimed to solve a reduced set of very specific tasks, usually involving a novel research topic. As a result, many of these tools were developed to have a standalone use and, thus, using and coordinating different tools is not possible without developing custom ad-hoc code in most of the cases [5]. Furthermore, up to the authors' knowledge, no tool is able to cope and cover all the tasks conforming to a KG life cycle, which are required for building and publishing KGs [5].

Consequently, building and publishing a KG becomes a complex and resource-consuming task that is not at the hands of all practitioners. On the one hand, practitioners must learn a wide spectrum of tools which some are research prototypes that are not suitable for a production environment, or their usability is hindered due to the lack of fundamental documentation. On the other hand, the fact that these tools can not be directly interconnected in order to work together

requires practitioners to develop ad-hoc pipelines to build and publish a KG [9–12]. Developing these ad-hoc pipelines has a high cost in time, personal resources and requires long cycles of debugging and maintenance, decreasing the project's productivity.

In this paper, a framework known as Helio is presented. The goal of the framework is to provide a tool that is able to perform all the tasks required for building and publishing a KG and, in case new functionalities are required, allows practitioners to integrate these without modifying the framework source code through independent plugins. To ensure its goal, Helio has been developed on top of a list of requirements that support the KG life cycle [13]. These requirements profile a system that is able to assist practitioners during the whole life cycle of a KG and that publishes the KG according to the Linked Data principles [4].

The Helio architecture has a modular design that, on the one hand, allows Helio to use some existing tools to perform these tasks and, on the other hand, allows practitioners to extend the framework in order to cope with new scenarios. In addition, Helio fosters the development of plugins for either using existing tools or implementing new functionalities since they are highly reusable. As a result, plugins prevent developing ad-hoc pipelines and allow other practitioners to cope with common scenarios without spending additional effort.

The Helio framework has been used in several contexts: A) European research projects from different domains, namely: VICINITY[1] (IoT in smart cities), BIMERR[2] (buildings and construction), DELTA[3] (energy demand response), AURORAL[4] (IoT in smart communities), and related research articles [14]; B) Research works [15–19], which relied on Helio to generate and/or publish their KGs; and C) Bachelor projects [20–23], in which Helio was extended or used to publish their results as a KG. Additionally, Helio has been presented in different tutorials [24, 25]. As a result, although Helio lacks formal experimental validation, its use in all these scenarios presents an indicator of its usability and usefulness.

The rest of this article is structured as follows: Section 2 reports the history, motivation, and a list of requirements on top of which Helio has been built; Section 3 introduces an analysis of proposals from the literature; Section 4 presents the framework design and its architecture; Section 5 provides a discussion about the framework introduced and how it meets the requirements elicited; Section 6 reports real-world cases where Helio has been used and; finally, Section 7 recaps our findings and conclusions.

[1] https://www.vicinity2020.eu/
[2] https://bimerr.eu/
[3] https://www.delta-h2020.eu/
[4] https://www.auroral.eu/

2. Requirements of a knowledge graph life cycle

Knowledge graphs have a well-defined and established life cycle [13], which is depicted in Figure 1. It consists of several steps, depicted as rounded boxes, which have one or more associated tasks that should be performed in each specific step, depicted as squared boxes in Figure 1. These tasks are usually related to one or more research problems, which are still active nowadays, or they lack a recommendation, fostering the numerous existing tools that tackle the same problem. As Figure 1 depicts, some of these tasks are also related to the Linked Data principles. The different steps of the life cycle are the following:

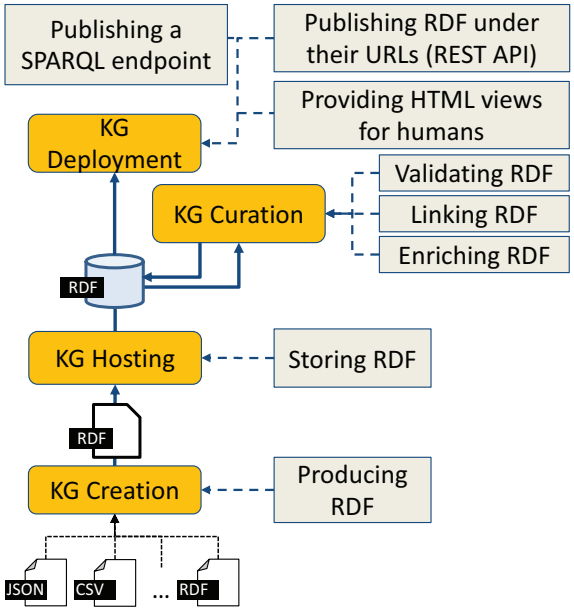


Figure 1. KG life cycle [13] and related tasks.

Knowledge graph creation: during this step, the knowledge graph is created by expressing its data as RDF [26] and according to an ontology. Sometimes, practitioners create the RDF data manually; however, in some cases, the RDF data is created using a materialiser [5]. Materialisers produce an RDF file by

fetching and translating data from heterogeneous data sources by means of translation mappings [6]. Practitioners rely on materialisers when the KG data must provide a homogeneous view of the data belonging to these heterogeneous data sources.

Knowledge graph hosting: this step aims at storing the RDF data in a suitable environment, which will allow interacting with such data during the following steps of the KG life cycle. For instance, during this step, data could be stored in a triple store or kept in memory depending on the user requirements. The RDF data to be hosted is usually provided manually by practitioners or uploaded automatically by an ad-hoc script; this is due to the fact that materialisers often only output an RDF file. The fact that the materialisers do not store the data automatically hinders the synchronisation of the hosted RDF data and the original one, which may change over time without updating the stored RDF one.

Knowledge graph curation: this step consists of several tasks: enriching, linking, and validating the RDF data. Enriching covers a large number of possible tasks [27], from transforming RDF data in order to increase its quality, e.g., removing white spaces or capitalising names, up to creating new data on the fly, e.g., completing the RDF data of a KG with machine learning [28]. Linking aims at producing links between the RDF resources by means of link rules [29]. Additionally, these links may involve RDF resources from different KGs; creating these links is one of the Linked Data principles [4]. Validating aims at ensuring that the RDF data follows certain restrictions, e.g., using the W3C recommended SHACL shapes [30]. As depicted by Figure 1 these tasks are executed over the stored RDF data and, although they are not mandatory, they improve the RDF data published afterwards.

Knowledge graph deployment: this step consists of publishing the RDF data of the KG for humans and/or machines so they can consume its content. For humans, the RDF data is usually presented embedded in a plain HTML document (lacking its formal semantics); instead, for machines, the RDF data is published at a resource level by means of a URL in a REST API that provides the RDF of such resource. A mixed solution, for humans and machines, proposed by the W3C consists of using HTML+RDFa documents [31], publishing HTML documents understandable by humans (with RDF injected within as plain HTML) that also contain RDF annotations inside that are understandable by machines. Additionally, for querying the RDF data, a SPARQL endpoint is usually provided [32].

As depicted by Figure 1, the different steps of the life cycle imply a set of related tasks that practitioners may perform using several of the different existing tools. Nevertheless, up to the authors' knowledge, these proposals usually focus on some tasks or steps, but they do not cover the whole KG life cycle [5]. To this end, a set of requirements have been elicited. These requirements find their origins mostly, but not uniquely, from real-world scenarios (like those presented in 6). These requirements profile a system that potentially covers the whole KG life cycle by implementing them. Furthermore, a system that implements these requirements builds and publishes the KG data according to the Linked Data principles, fostering the good practices promoted by the W3C. The requirements are the following:

- (KG Creation) R01: The system allows practitioners to provide as input an RDF file, in all likelihood created manually, for feeding the life cycle.
- (KG Creation) R02: The system provides a materialisation tool to translate the heterogeneous data, i.e., non-RDF, from a set of heterogeneous data sources into RDF.
- (KG Creation) R03: The materialisation tool of the system understands more than one mapping language, reducing the chances for users of needing to learn a new mapping language or providing bespoke features of such mapping language missing in other [33].
- (KG Creation) R04: The materialisation tool of the system relies on a mapping language that allows expressing a set of functions, and the tool implements these functions. This allows practitioners to use these functions to clean data before translating it into RDF.
- (KG Creation) R05: The materialisation tool of the system allows defining link rules and applying such rules for linking resources that belong to the RDF data of the KG.
- (KG Creation) R06: The system provides a mechanism to use other existing materialisation tools. This allows practitioners to use a materialisation tool known by them and, therefore, not need to learn a new mapping language.
- (KG Creation) R07: The system provides reusable extension mechanisms that allow extending the provided materialisation tool and other system

features in order to cope with new scenarios without forcing practitioners to develop ad-hoc software.
- (KG Hosting) R08: The system provides different configurable options for storing the RDF data [5]. For instance, the system may be configured to store in-memory RDF data for quick retrieval.
- (KG Hosting) R09: The system provides mechanisms to synchronise the stored RDF data generated by one or more materialisation tools and the original heterogeneous data.
- (KG Curation) R10: The system allows practitioners to use existing tools that aim at enriching, validating, or linking RDF data that has been previously stored by the system.
- (KG Deployment) R11: The system provides a REST API that publishes each resource in the RDF data through its URI using either the HTTP or HTTPS protocol.
- (KG Deployment) R12: The system provides a SPARQL endpoint according to the W3C specification [32]. As a result, the system allows practitioners to query the RDF data of the KG.
- (KG Deployment) R13: The system provides content negotiation so practitioners can consume the RDF data and/or the SPARQL results in different serialisations.
- (KG Deployment) R14: The system publishes HTML views for assisting practitioners during data consumption.
- (KG Deployment) R15: The system provides mechanisms to customise the HTML views, e.g., to allow practitioners to change the aesthetics of the HTML views.
- (KG Deployment) R16: The system provides mechanisms to customise and embed meta-annotations in the published HTML views. As a result, the system allows transforming the plain HTML into HTML+RDFa [34].

Notice that the previous requirements not only describe a system that is able to assist practitioners during the whole life cycle of a KG. In addition, a system that implements all these requirements publishes the RDF data of a KG according to the Linked Data principles [4], namely: 1) Use URIs as names for things covered by R01 and/or R02; 2) Use HTTP URIs so that people can look up those names, covered by R11; 3) When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL), covered by R11 and R12; and 4) Include links to other URIs, so that they can discover more things, covered by R10.

3. Related Work

There are a wide number of tools from the literature that have been designed to address specific tasks or steps from the KG life cycle depicted by Figure 1. However, up to the authors' knowledge, none is able to cope with the whole KG life cycle; as also pointed out by Simsek et al. [5]. In this section, the different tools are analysed from the point of view of the requirements elicited by section 2 and the step, or steps, of the KG life cycle that they address.

Table 1 shows the different categories of these tools from the literature. Additionally, Table 1 reports which of the requirements are covered by all the tools from the category (✓), or they do not cover (-), or are partially covered by some of the tools (∼). The categories and their tools are following surveyed.

3.1. Knowledge graph creation

RDF materialisation is an approach widely used to generate the RDF data of a KG from a set of heterogeneous sources that counts with a large number of existing tools [14, 35–39]. Although some of these may differ on efficiency or suitability when applied in certain contexts, in general, they have the same workflow. First, a practitioner manually writes a set of translation mappings; then, these mappings are provided as input for the materialiser; finally, the materialiser fetches and translates the data producing the RDF data that is written in a file. As a result, since these tools provide as output an RDF file, they only cover the KG Creation step from the life cycle, i.e., excluding any requirement from R08 and forth.

Although all the materialisers implement the requirement R02, not all of them implement the other KG Creation requirements: only a few materialisation tools are able to understand more than one translation mapping (R03); a large number of materialisers are able to apply functions when translating heterogeneous data into RDF. However, they are namely meant for cleaning data rather than linking RDF resources (R04 and R05); these tools are developed to translate heterogeneous data and, thus, some of them are not able to take data already in RDF as input (R01). Finally, up to the authors' knowledge, none of these tools is designed to work in combination with another materialisation tool (R06), nor provide extension mechanisms to cope with new scenarios (R07), e.g., a new format from which to translate data into RDF.

KG life cycle	Requirements	Categories of tools from the literature: RDF materialisers	OBDI/A	RDF frameworks	RDF triple stores	RDF publishers
KC Creation	R01	∼	-	∼	-	-
	R02	✓	-	∼	-	-
	R03	∼	∼	∼	-	-
	R04	∼	∼	-	-	-
	R05	∼	-	-	-	-
	R06	-	-	-	-	-
	R07	-	-	-	-	-
KG Hosting	R08	-	-	∼	-	-
	R09	-	-	-	-	-
KG Curation	R10	-	-	∼	∼	-
KG Deployment	R11	-	-	∼	∼	-
	R12	-	∼	∼	✓	∼
	R13	-	-	∼	✓	∼
	R14	-	-	-	-	∼
	R15	-	-	-	-	∼
	R16	-	-	-	-	-

Table 1

Elicited requirements met by existing tools types.

Ontology Based Data Integration (OBDI) and Ontology-based Data Access (OBDA) tools are used when there is a non-RDF database with large amounts of data for which materialisation proposals fall short [40]. These tools focus on providing a SPARQL endpoint and translating the SPARQL queries received into one or more languages. OBDA are tools that only translate from SPARQL to just one language [41–46], instead OBDI tools that are able to translate a SPARQL query into other multiple languages at once [47–49]. Since these tools are not actually building a KG, they do not cover any requirement from the KG Creation step from the life cycle. Instead, these tools cover some from the KG Deployment, especially those related to SPARQL, since they allow consuming the heterogeneous data from the databases by means of SPARQL queries.

OBDI and OBDA tools allow answering SPARQL queries over data from heterogeneous databases as if a KG with such RDF data would exist. However, some of these tools expect the queries to be provided programmatically rather than through a published SPARQL endpoint (R12), and some of these tools only support SELECT queries or SELECT without special statements like FILTER; which may be a serious limitation when querying the data of a KG. Finally, since these tools perform a translation of queries, although SPARQL supports functions that could be used for cleaning or linking, not all the tools are able to cope with such functions during query translation (R03 and R04). Furthermore, since these tools only translate queries instead of building and publishing RDF data, they do not cover the rest of the requirements from R08 and further, with the exception of R12 for some tools.

3.2. Knowledge graph curation

KG Curation involves a large number of tools from the literature that can be divided into four categories: RDF enriching tools [50], RDF link discovery tools [7], RDF validation tools [51], and RDF quality assessment tools [52]. Nevertheless, this categorisation is far from complete since the number of tools that may fall in this topic is numberless. These categories group the tools that authors consider the most common to appear in the KG life cycle, but others could also be considered as related to knowledge graph curation and do not fall in any of the previous categories.

Notice that KG Curation is a step that occurs once the RDF data has been stored after the KG Hosting step. It is worthwhile to mention that the tasks involved in KG Curation are not necessarily blocking for those happening in KG Deployment, entailing that they are fully optional. In fact, any of these tools could be used by a system covering a KG life cycle as stated by R10; however, by themselves, they do not cover any requirement of the elicited ones. Due to this fact, the tools analysed in this subsection are not included in Table 1.

RDF enriching tools have a wide number of goals. For instance, some tools aim at completing with new

information existing RDF data [27] whereas others aim at summarising existing RDF data [53].

RDF link discovery aims at producing relationships between local RDF resources and other RDF resources allocated in different KGs [7]. On the one hand, there is a wide number of tools that aim at producing link rules [54–62], i.e., restrictions under which two RDF resources are linked. On the other hand, other tools focus on applying those rules efficiently and producing the links among resources [63–65].

RDF validators are tools that specify whether data expressed in RDF conform to a set of restrictions. There is a specification for expressing these restrictions that is a W3C standard, i.e., SHACL [30], and other nonstandard specifications [66]. These tools usually take as input an excerpt of RDF data and a set of restrictions and produce a validation report.

RDF quality tools aim at investigating and quantifying the quality of a KG and the parameters influencing such quality [67]. To this end, the literature counts with proposals of different nature: from tools [68] to metrics [52], and other approaches [69].

3.3. *Knowledge graph hosting and deployment*

RDF frameworks aim at providing practitioners with several functionalities, e.g., pragmatically choosing different environments where to host their RDF data that belong to different steps of the KG life cycle [70–74]. Some tools like Star Dog[5] (that is also a triple store) implement a wider number of requirements related to different steps of the KG life cycle (R01, R02, R03, or R10 among others). Some others, like Jena [70], are also suitable for validation (R10). Others, like RDF4J[6] focus on providing mechanisms to practitioners to choose from different triple stores where to allocate their RDF data (R08).

In most of the cases, the KGs are deployed by storing their RDF data into triple stores [75–79]. These stores host RDF data and provide a SPARQL endpoint (R12). Some triple stores also publish each resource under a URL (R11), and others implement content negotiation for the SPARQL endpoint and RDF resources (R13), including HTML documents. Furthermore, some triple stores provide curation techniques, e.g., SHACL validation. Nevertheless, triple stores are not suitable for any other task within the KG life cycle.

[5]https://www.stardog.com/
[6]https://rdf4j.org/

RDF publishers aim at providing human interfaces (HTML) for those SPARQL endpoints and resources published exclusively with machine interfaces. Some tools, like YASGUI [80], publish a human SPARQL interface for a given SPARQL endpoint (R12 and R13). Others, like Pubby [81], also publish human interfaces for the resources provided by the SPARQL endpoint (R14). Some others, like Elda[7], allow customising the aesthetics of HTML views meant for humans relying on templates (R15). Nevertheless, up to the authors' knowledge, none of these tools allows the customisation of HTML views for transforming them into HTML+RDFa [34] (R16).

4. The Helio Framework

Helio is a framework built to meet the requirements previously elicited and explained. The goal of Helio is to build a KG from heterogeneous data sources (which may include RDF sources) and publish the KG data following the Linked Data principles. In order to meet all the requirements, the Helio framework is divided into four logic modules, each of which aims at implementing a set of the elicited requirements. These modules are implemented as Java artefacts, although they could be implemented or viewed as microservices alternatively. The modules and the requirements that they cover, depicted in Figure 2, are the following:

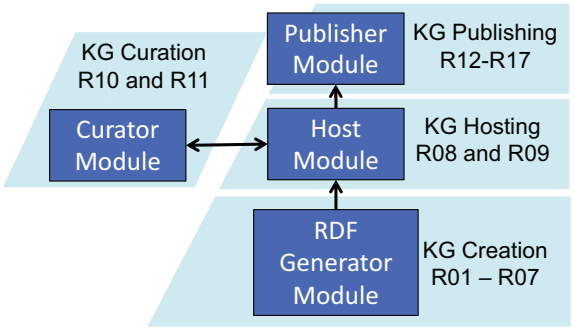


Figure 2. Helio framework

RDF Generator Module[8]: this module focuses on generating the RDF data of a KG from a set of heterogeneous data sources, including RDF data sources. The output of this module is one or more fragments

[7]http://epimorphics.github.io/elda/index.html
[8]https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users

of RDF data that are passed to the Hosting Module. This module aims at covering the requirements related to KG Creation, namely, R01 to R07. Subsection 4.1 provides a detailed description of this module.

Hosting Module[9]**:** this module focuses on how the RDF data of a KG is stored and provides a SPARQL interface for interacting with it. It allows the RDF Generator Module to store data, the Curation Module to update existing data, and the Publisher Module to read the stored data. This module aims at covering the requirements related to the KG Hosting, namely, R08 and R09. Subsection 4.2 provides a detailed description of this module.

Curation Module: this module focuses on the different tasks related to the curation of RDF data for enriching the KG. This module interacts with the Hosting Module through the SPARQL endpoint for reading the current data, applying a curation technique (e.g., data linking), and storing the generated RDF data again in the Hosting Module. This module aims at covering the requirements related to the KG Curation, namely, R10. Subsection 4.3 provides a detailed description of this module.

Publisher Module[10]**:** this module focuses on the publication of the RDF data of a KG. This module publishes different views of the data. On the one hand, it implements a REST API to access the RDF resources whose format may be chosen relying on content negotiation. On the other hand, the module also publishes HTML views that can be customised into HTML+RDFa, which can be retrieved using content negotiation. Additionally, a standard SPARQL [32] endpoint and the whole dataset are also published for either querying the data or downloading a dump of the dataset (also available in different formats). This module aims at covering the requirements related to KG Publishing, namely, from R11 to R16. Section 4.4 provides a detailed description of this module.

In the following subsections, these modules are explained in detail, providing an insight view of their implementation. Then, in section 5 it is explained how the framework allows publishing KGs according to the Linked Data principles and how it covers the elicited requirements.

[9]https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#repositories

[10]https://github.com/oeg-upm/helio/wiki/Helio-Publisher

4.1. RDF Generator Module

The RDF Generator Module is in charge of generating the RDF data of a KG and providing this data to the Hosting Module. In order to fulfil its goal, this module is built upon two generic components that must be instantiated in an implementation, i.e., data providers and data handlers, and a component to translate data into RDF if required, i.e., the data translator. The details of the translation process are specified in an Helio bespoke mapping language, the conceptual mapping, that must be provided to the RDF Generator Module as input. Additionally, there is the last component named resources orchestrator that organises the whole translation process and pushes the generated RDF data into the Hosting Module.

The data providers are components in charge of retrieving data from one data source. These components are agnostic to the format of the data; its only goal is to deal with the protocols for retrieving the data. After a data provider obtains the data, such data is passed to the data handlers. The current Helio implementation counts with several data provider instantiations[11]. *Example 1: the URLProvider is able to retrieve data from a URL based on several protocols such as HTTP, HTTPS, FTP, or file. Nevertheless, the URLProvider is agnostic from the format of the data retrieved.*

The data handlers are components that focus on fetching fragments of information from the data provided by a data provider. These components are highly related to the format of the data since they need to iterate or access specific positions of the data in order to fetch the fragments. Notice that they are totally agnostic to the protocols involved for retrieving such data. The current Helio implementation counts on several data handlers instantiations[12]. *Example 2: assuming that the data retrieved in Example 1 was a JSON file, the JsonHandler should be used for iterating over the file and retrieving different values by means of JSONPath expressions.*

The RDF translator takes as input a conceptual mapping that specifies what data providers shall be used and to which data handlers they have to pass the retrieved data. In addition, these mappings hold a set of translation rules that are related to the data handlers. The rules usually contain some filtering expressions

[11]https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#data-providers

[12]https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#data-handlers

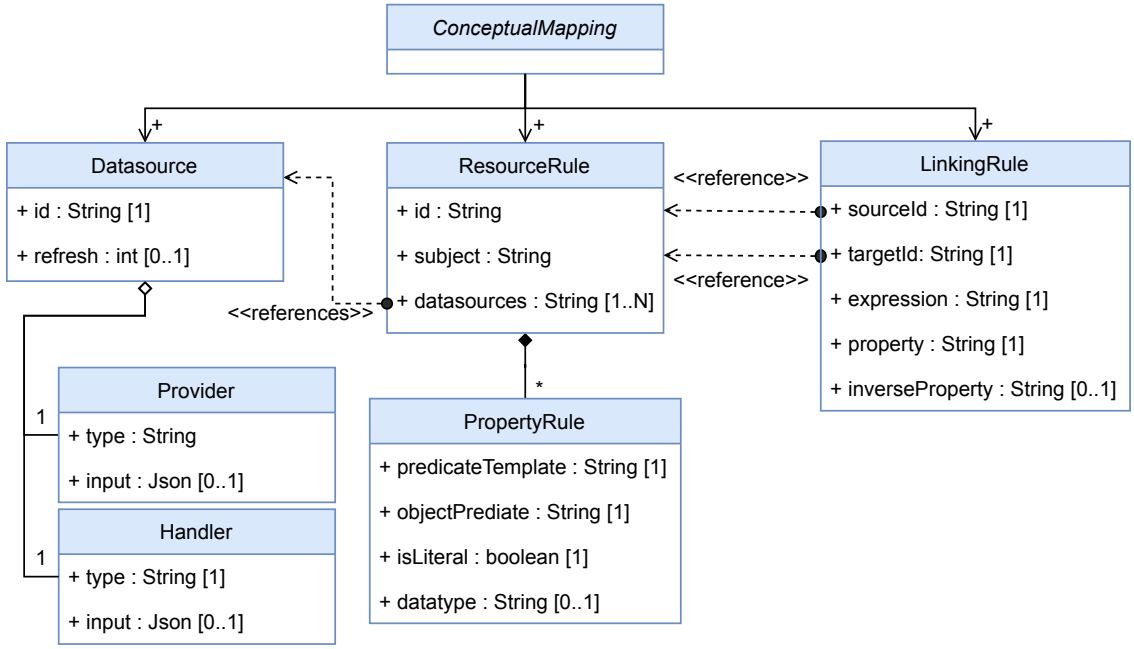


Figure 3. Helio Conceptual Mappings model

and optionally cleaning functions that require fetching fragments of information from the data retrieved by the data providers by using their related data handlers to process the filtering expressions. Additionally, the conceptual mappings may include some linkage rules to be applied after the translation. Finally, once the RDF translator has been initialised with a conceptual mapping, this component initialises and connects the different data providers with their respective data providers and then remains on standby. In case the RDF translator would be provided with valid RDF data, i.e., no translation is required since the data handler is meant for RDF, this component will automatically provide the RDF data as a result.

The resources orchestrator is the component that triggers the RDF translator when required on-demand by any other module. At that moment, the resources orchestrator invokes the RDF translator that will generate the RDF data. Then, the resources orchestrator pushes this data into the Hosting Module. Alternatively, if specified in the conceptual mappings, the resources orchestrator can trigger the RDF translator periodically instead of on-demand.

Notice that anytime the RDF data is pushed to the Hosting Module, this module could handle it in different ways. For instance, if an excerpt of RDF is already present in the triple store, it could replace the old triples with the new ones. On the other hand, the Host Module could store the triples in different named graphs, and anytime an excerpt is given, it would be stored into a new named graph, not overriding the old version of such an excerpt (this would be especially interesting for versions of the data or even historical data).

4.1.1. Conceptual Mappings

The Conceptual Mappings[13] specify how the translation of data is performed. As depicted by Figure 3, a *ConceptualMapping* is conformed by three main elements: one or more *Datasource*, one or more *ResourceRule*, and one or more *Linking Rule*.

A *Datasouce* describes a source of data, which has a unique identifier (*id*) and a refreshing time (*refreshTime*) that specifies whether the generation of data is performed on-demand (if null) or periodically. In addition, a *Datasource* counts with two other elements: a data *Handler* and a data *Provider*.

A *Provider* and a *Handler* have a *type* that refers to a specific instantiation, for instance the name of a class such as *JsonHandler* or *URLprovider*, and an input that must be a JSON document for configuration, for instance for the *URLprovider* the URL from where the data must be fetched.

[13] https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#helio-mappings

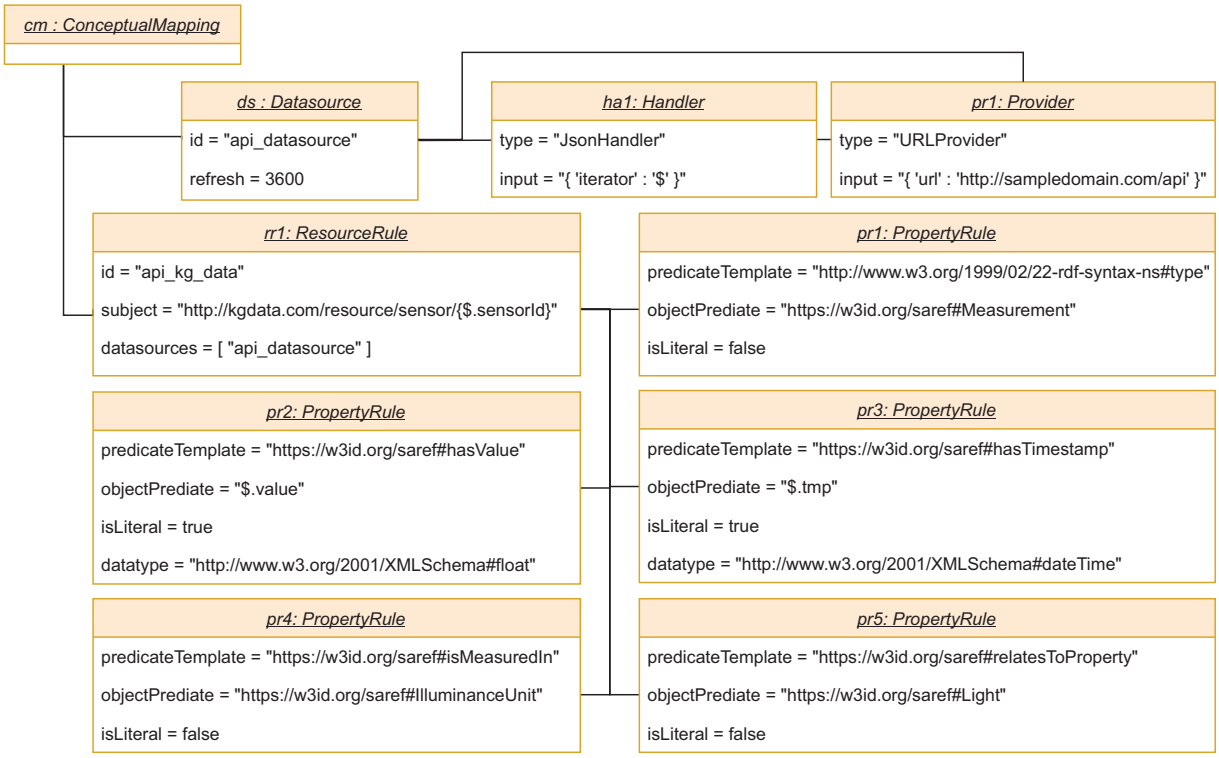


Figure 4. Example of Conceptual Mappings for a REST API

A *ResourceRule* describes how the data from one or more sources is translated into RDF. It has a unique identifier (*id*), a set of *Datasource* identifiers (*datasources*), and a *subject* that specifies how the subject of a set of triples is generated. Additionally, a *ResourceRule* is related to zero or more *PropertyRule*; each of which specifies how to generate a predicate and an object related to the former subject (*predicateTemplate* and *objectPrediacte*, respectively), and also, if the object is a literal (*isLiteral*) or the datatype of such literal (*datatype*).

A *LinkingRule* describes how to link resources (the subjects) from the RDF generated with the rules of two *ResourceRule*. The *LinkingRule* has two *ResourceRule* identifiers, one related to the subject of the link (*sourceId*) and one related to the object of the link (*targetId)*. Also, the *LinkingRule* has an *expression* that is a link rule [61], an RDF predicate to relate both subjects (*property*), and a predicate that will be generated in inverse order *inverseProperty* (linking the target subject with the source subject).

Notice that a *LinkingRule* can only relate RDF resources within the KG generated by the framework. Linking the RDF resources from different KGs is out of the scope of these linking rules. Therefore, they must not be considered as a curation technique.

Figure 4 depicts a simple Conceptual Mapping instantiation specifying how to integrate data from a REST API that publishes JSON data about sensors that measure luminance. A sample payload is the following:

```
{
  "sensorId" : "001",
  "value" : "3.2",
  "tmp" : "2005-10-30T10:45:00Z"
}
```

Notice that the Conceptual Mappings are data structures that the RDF Generator component handles internally. This entails that the input provided to this component may have different serialisations that are translated into this data structure internally. For instance, the Conceptual Mapping depicted in Figure 4 can be the result of translating an equivalent mapping from a JSON serialisation (as shown in Appendix A), or translating an equivalent mapping from RML (as shown in

Appendix B), or from a WoT-Mapping[14] (as shown in Appendix C).

The RDF Generator Module has several internal translators in order to understand different mapping languages, like RML[15], the WoT-Mappings, or the JSON serialisation[16] of the model depicted by Figure 4.

4.1.2. Extending the RDF Generator Module

As it has been explained, the RDF Generator Module is capable of generating RDF from heterogeneous data sources, cleaning the data, and linking the RDF resources generated. Although it counts on several data providers and handlers for achieving this task, new scenarios may introduce protocols or formats currently not supported by the module.

For this reason, the RDF Generator Module counts with a dynamic system for loading plugins[17]. This allows users to develop new data providers or handlers without modifying the code of Helio, and load these extensions dynamically.

As an example, in the repository of plugins[18] it can be found a data handler that allows Helio to fetch data from an Ethereum blockchain[19]. A new user that would like to use this plugin should only download the jar, configure Helio to use it[20], and define a correct mapping[21].

Notice that the RDF translator component is capable of dealing with data that is already expressed in RDF. Therefore, a new data provider that relies on existing materialisation techniques could be implemented. This provider would receive as input the mappings understandable for that technique, and the technique would be invoked as a regular data provider. It is worth mentioning that, similarly to materialisers, OBDI or OBDA techniques could also be included as data providers.

As a result, thanks to the plugin system, the RDF Generator Module is capable of reusing code and prevents the generation of non-reusable ad-hoc pipelines. Additionally, although it generates RDF from heterogeneous sources, it could be used with plugins that rely on third-party techniques for RDF generation.

4.2. Hosting Module

This module publishes a SPARQL endpoint for the other modules to store, read, or update RDF data. For this goal, the current Helio implementation relies on SAIL Configurations that enable a user to specify where to store RDF data. For instance, the following configuration stores the data in an existing triple store.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rep: <http://www.openrdf.org/config/repository#>.
@prefix sparql: <http://www.openrdf.org/config/repository/sparql#>.

[] a rep:Repository ;
   rep:repositoryID "example" ;
   rdfs:label "SPARQL endpoint at http://example.org/" .
   rep:repositoryImpl [
      rep:repositoryType "openrdf:SPARQLRepository" ;
      sparql:query-endpoint <http://example.org/sparql> ;
      sparql:update-endpoint <http://example.org/sparql/update> ;
   ];
```

Instead, the configuration below specifies that the triples must be stored in the file system.

```

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rep: <http://www.openrdf.org/config/repository#>.
@prefix sr: <http://www.openrdf.org/config/repository/sail#>.
@prefix sail: <http://www.openrdf.org/config/sail#>.
@prefix ms: <http://www.openrdf.org/config/sail/memory#>.
@prefix sb: <http://www.openrdf.org/config/sail/base#>.

[] a rep:Repository ;
   rep:repositoryID "example" ;
   rdfs:label "Example Memory store" ;
   rep:repositoryImpl [
```

[14] http://iot.linkeddata.es/def/wot-mappings/index-en.html
[15] https://github.com/oeg-upm/helio/wiki/Streamlined-use-cases#materialising-rdf-from-csv–xml-and-json-files-using-rml
[16] https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#helio-mappings
[17] https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-developers#helio-plugins
[18] https://github.com/oeg-upm/helio-plugins
[19] https://github.com/oeg-upm/helio-plugins/tree/master/providers/ethereum-provider
[20] https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#advanced-configuration
[21] https://github.com/oeg-upm/helio-plugins/tree/master/providers/ethereum-provider#ethereumconnector-example-mapping

```
    rep:repositoryType "openrdf:
        SailRepository" ;
    sr:sailImpl [
        sail:sailType "openrdf:MemoryStore"
            ;
        sail:iterationCacheSyncThreshold "
            10000";
        ms:persist true;
    ]
].
```

The Hosting Module is configured with one of these SAIL Configurations and then publishes a SPARQL endpoint for the rest of the modules to be used. Notice that this flexibility allows users to adapt to certain scenarios where the computational resources are limited (e.g., deploying Helio in a Raspberry Pi board) and thus, choosing a suitable environment becomes paramount.

4.3. Curation Module

The Curation Module aims at performing different curation tasks, for instance, linking resources or completing RDF triples. Therefore, this module can have one or more implementations depending on the task at hand.

The current Helio framework allows for any Curation Module implementation to interact with the rest of the framework by relying on a standard SPARQL interface. These implementations must access the generated data by means of the Hosting Module, which publishes the SPARQL endpoint, perform the desired curation task, and then store the output by using the Hosting Module through the SPARQL endpoint. As a result, the RDF of the KG published by the Publisher Module will include these modifications.

Notice that this mechanism allows any user to define a service to perform a specific curation task; its only requirement is to interact with the Hosting Module through a SPARQL 1.1 interface. As a result, this service could be reused by any third-party entity that will have to deal with a similar, or the same, curation challenge.

Notice that the current Helio implementation allows linking RDF resources that are the result of translating data from heterogeneous sources. In other words, Helio does not implement a linker among different KGs (which would be a Curation Module), only among the resources of a local KG generated by Helio.

4.4. Publisher Module

The Publisher Module is in charge of making the RDF data from the Hosting Module available through the HTTP protocol, i.e., it publishes a REST API for consuming the data. The current implementation of the Publisher Module is a Spring Boot Java service.

The data from the Hosting Module is published by this module at three levels: RDF resource level, when the URL of a specific existing RDF resource is requested the Publisher Module outputs its triples; SPARQL level, the Publisher Module enables a standard SPARQL 1.1 endpoint for querying all data stored by the Hosting Module; Dataset level, the Publisher Module provides a dump containing the triples that conform the dataset stored in the Hosting Module.

The Publisher Module implements content negotiation by means of HTTP headers that enable consuming any of the data published in different formats. For instance, the module will provide a client with an HTML view if a request with the *text/html* is performed; instead, if the same request uses *text/turtle* the same data will be output in raw RDF turtle. Figure 5 from Annex D shows the standard HTML views that the Publisher Module provides for its SPARQL endpoint (implemented with YASGUI [80]) (shown by Figure 5a), any RDF resource (shown by Figure 5b), and the dataset (shown by Figure 5c).

Besides the standard views of RDF resources (shown by Figure 5b), the Publisher Module implements a mechanism to customise the HTML views of the resources, as depicted by Figure 5d from Annex D. It allows associating the URLs of these resources to a specific HTML file in which the information is dynamically injected. As a result, a user can customise the HTML views of the resources. Furthermore, these views can also include RDFa annotations.

Finally, the Publisher Module allows defining dynamic views that are HTML documents in which the data injected is the result of a SPARQL query. In other words, a user can choose a subset of data from the Hosting Module by means of a SPARQL query and associate to such view a URL that does not exist in the dataset. Nevertheless, if a client requests such URL to the Publisher Module, the module will automatically fetch the data and inject the result into a customised HTML view previously provided. A sample of this kind of usage is available at https://helio.vicinity.iot.linkeddata.es/tests/vicinity-wot; furthermore, the view of this example is HTML+RDFa.

4.5. Communication between modules

The communication among the different Helio modules depicted in Figure 2, as well as the nature of the information exchanged and the way this is done, is heavily related to specific implementations of the framework.

Currently, the RDF Generator Module of the framework is distributed as a Java dependency. The Publisher Module is distributed as a Spring Boot Service that imports the RDF Generator Module; therefore, they communicate through the interfaces of the Java library described by the Helio framework repository[22]. Both components also interact with the Host Module, which currently is implemented with RDF4J and allows deploying an embedded triple store (and therefore the communication among components is performed programmatically), or allows to use a remote SPARQL endpoint[23] (in which case the communication is performed through SPARQL queries).

Finally, any Curator Module can interact with the Host Module through the SPARQL endpoint when such module is implemented using a remote triple store. Notice that if the Host Module is implemented as an embedded RDF4J triple store, the possibility of using Curator Modules is not possible. Nevertheless, having an embedded triple store is suitable for some scenarios, as mentioned in Annex E.

Notice that the modularity of the Helio framework suits a microservice-based implementation, which is one of the future aims of the authors. In such an implementation, the communication among the components will be totally different from the current one.

5. Discussion

This section aims at providing a discussion divided into several subsections. The former explains how the framework provides the means for publishing a KG following the Linked Data principles. The latter explains how the framework meets the requirements elicited in Section 2. Finally, a brief explanation of the use cases is given.

[22] https://github.com/oeg-upm/helio-framework

[23] https://github.com/oeg-upm/helio/wiki/Streamlined-use-cases#using-a-triple-store-with-a-sparql-endpoint-as-helio-repository

5.1. Enabling Linked Data principles

The Linked Data principles establish good practices that must be followed when publishing a KG [4]. These principles are namely:

- **P1:** Use URIs to identify things.
- **P2:** Use HTTP URIs so that people can look up those names.
- **P3:** When someone looks up a URI, provide useful information using the standards (RDF, RDFS, SPARQL).
- **P4:** Include links to other URIs so that they can discover more things.

The Helio framework enables **P1** by allowing users to provide valid RDF data (that must identify things with URIs) or providing mechanisms to translate heterogeneous data into RDF.

P2 and **P3** are enabled due to the fact that the framework publishes over HTTP the RDF resources identified by URIs; in addition, when these URIs are looked up, they provide the information of those resources by means of standards (RDF, HTML, HTML+RDFa, SPARQL).

Finally, the framework allows generating links among the RDF resources of the same dataset thanks to the linking rules supported by the Generator Module. Nevertheless, other linking techniques can also be used as a Curation Module implementation. As a result, the framework enables the **P4** principle.

Notice that none of the tools analysed in the literature explained in Section 3 allowed users to fully follow the Linked Data principles. In order to follow them, a user should rely on several of the analysed tools.

5.2. Requirements met

The requirements elicited in section 2 are grouped by the KG step. Similarly, the modules of Helio are split into the same steps, easing the coverage analysis of these requirements.

The Generator Module meets the requirements related to KG Creation, i.e., **R1-R07**. This module produces RDF data by either translating a set of heterogeneous sources (**R02**) or by allowing users to provide data already in RDF (**R01**). Furthermore, the translation of the data can be specified using different mapping languages (**R03**) as shown in Annexes A, B, or C. In the particular case of using the native language of the framework, the conceptual mappings allow defin-

ing cleaning functions (**R04**) or linking rules (**R05**). Finally, the Generator Module implements a plugin-based system that prevents users from developing ad-hoc code that is not reusable (**R07**). Relying on this plugin-based system existing materialisation tools can be used as a *DataProvider* to translate heterogeneous data into RDF instead of using the Helio native translation component (**R06**).

The Hosting Module allows, by means of an RDF SAIL configuration, to choose different environments to store the data from an existing triple store to disk-based persistence (**R08**). The data stored in these environments is provided by the Generator Module (**R09**), which can be configured using conceptual mappings for pushing the data synchronously on-demand or asynchronously periodically each quantum of time.

The Host Module allows plugging any Curator Module (**R10**), thanks to the architecture of the framework. Depending on the RDF SAIL configuration, the Host Module publishes a SPARQL endpoint with which the Curator Modules can interact, or, alternatively, they should pragmatically interact with the Host Module.

Finally, the Publisher Module meets the requirements related to the KG Deployment, i.e., **R11-R16**. The publisher provides a REST API for accessing the RDF resources (**R11**), performing queries using a standard SPARQL endpoint (**R12**), and downloading the KG dump. Furthermore, relying on HTTP content negotiation, the RDF resources, or the dump, are provided in different formats (**R13**). For HTML MIME types, the Publisher Module already provides a set of default views (**R14**). Nevertheless, it also allows defining custom HTML views (**R15**) that may include meta-annotations transforming HTML into HTML+RDFa (**R16**).

As a final remark, notice that none of the tools analysed in Section 3 was able to meet all the requirements elicited in Section 2. Up to the authors' knowledge, the Helio framework is the first tool to meet all these requirements, allowing the users to cover the whole life cycle of a KG.

5.3. Architecture details & Use cases

The Helio framework is built upon the different modules shown by Figure 2. Although in principle, all these modules are paramount for the framework, not all of them are necessary to exist in a real-world deployment. To this end, the use cases identified for the framework are described in Annex E. As a result, it is shown how the combination of either the Publisher Module and the RDF Generator Module with the Host Module is always required. Instead, the Curator Module is always an optional module that is not required.

6. Helio framework adoption

The Helio framework lacks formal experimentation. Nevertheless, it has been widely used in different contexts. The wide adoption of the framework is an indicator of its usability and usefulness.

VICINITY[24]**:** in the European project VICINITY, Helio was used to implement the Gateway API Query Distributed component. The goal of this component was to answer SPARQL queries using the data from a set of REST endpoints publishing JSON data about sensors, which is highly dynamic. Helio was extended in this project in order to understand the WoT-Mappings developed for this project [18], in which ontology is publicly available[25]. In the context of this project, this component was the heart of the semantic interoperability approach responsible for allowing components to transparently exchange and understand data. In this context, Helio was extended with the WoT-Mappings translator.

As a result of this project, a standalone proposal called eWoT that enables semantic interoperability for ecosystems of sensors was released [14]. This proposal relies on Helio to perform the translation on the fly of the data required to answer an issued query.

DELTA[26]**:** in the European project DELTA, Helio implements the semantic interoperability architecture [19] for exchanging Demand Response orders. In this project, Helio was part of the Common Information Model (CIM)[27] software that enables components to exchange data in different formats. Then the CIM offers services to exchange data (by translating the heterogeneous data on the fly into RDF expressed according to the DELTA ontology [82]), validate, and publish such data.

Additionally, in this project, Helio was used to publish JSON data stored in an Hyperledge Blockchain as RDF, allowing users to easily consume such data. In this context, Helio was extended for including custom

[24] https://www.vicinity2020.eu/vicinity/
[25] http://iot.linkeddata.es/def/wot-mappings
[26] https://www.delta-h2020.eu/
[27] https://github.com/oeg-upm/DeltaCimApp

templates for publishing the data and enabling a validation mechanism to ensure data quality.

AURORAL[28] in the ongoing European project AURORAL, Helio is going to be used for translating the data published by a large number of data sources, such as IoT devices and services, into RDF and for combining this output with Thing Descriptions from the Web of Things (WoT) standard to enhance the current WoT discovery specification [83] by allowing not only to discover resources using static data (provided in the Thing Descriptions) but also taking dynamic data into account (coming from the data sources).

COGITO[29]**:** in the ongoing European project COGITO, Helio is going to be used to generate and publish KGs for Digital Twins from heterogeneous data sources to build their virtual model.

OntoCommons[30]**:** in the ongoing European project OntoCommons, Helio has been used exclusively to publish a data portal [31] that exposes a list of ontologies and, for each one, their related information. The views for the listing of the ontologies and the information of each one have been customised instead of relying on the Helio default views.

Astrea [17]: this project aims at automatically generating SHACL shapes automatically from a set of input ontologies. Helio is used to integrate and publish data from different sources, which conforms to a Knowledge Graph that is the pillar component for the automatic generation of the shapes. The Helio endpoint with the integrated data is public available[32].

Themis [16]: this project deals with ontology testing; it generates conformance reports for an ontology taking as input a set of test cases provided by a user. In this case, Helio was used to publish the conformance reports with custom HTML templates that have embedded RDF, i.e., HTML+RDFa. The endpoints published with Helio are accessible from the main page of Themis[33].

Semantic Blockchain: Helio has set the pillars to develop a research line that aims at combining semantic web technologies such as SPARQL, RDF, or ontologies with blockchain. This line currently counts with two papers [15, 84] that focus on studying the feasibility of storing RDF directly in the blockchain or storing JSON and using Helio to publish the data (allowing resource and query access). In this context, two plugins have been developed for Helio that are the Ethereum and Hyperledge connectors.

This research line was originated from a master thesis [20] in which the feasibility of using Helio to publish data stored in a blockchain regardless of its implementation (e.g., Bitcoin, Ethereum, or Hyperledge) was studied and analysed.

Bachelor works: Helio has been used in two bachelor works developed at the Universidad Politécnica de Madrid. The former work aimed at implementing a smart office in which several sensors had to gather data and control their values to ensure that the work environment fulfilled a set of health KPIs proposed by the European Commission [21]. Helio was deployed in a Raspberry Pi board, and its goal was to fetch the data coming from a set of sensors, which were pushing their data into an MQTT broker. For this purpose, an MQTT connector for Helio was developed.

The latter Bachelor's work aimed at studying factors that were related with the *rabies virus* propagation [22]. For this purpose, Helio integrated a file with data endowed by the student and several sources of information, namely: the PanTHERIA database[34], some data from the Encyclopedia of Life[35], and Wikidata.

NLP: in the context of the Natural Language Processor, Helio is currently used to assist Name Entity Recognition (NER) tasks. For this purpose, the Valkyr-IE plugin has been developed, which allows extracting entities from a set of given texts using the tool Valkyr-IE[36].

Lectures: currently, Helio is being used to support different courses related to semantic web and knowledge graphs imparted at the Universidad Politécnica de Madrid.

7. Conclusions and Future work

In this article, the Helio framework for building and publishing KGs as Linked Data has been presented. The framework sets its pillars on top of several requirements that establish the life cycle of KGs, meeting these requirements and allowing practitioners to publish KGs following the Linked Data principles. Furthermore, the framework counts with a plugin system

[28] https://www.auroral.eu
[29] https://cogito-project.eu/
[30] https://www.ontocommons.eu/
[31] https://data.ontocommons.linkeddata.es/index
[32] https://astrea.helio.linkeddata.es/
[33] http://themis.linkeddata.es/catalogue.html
[34] https://ecologicaldata.org/wiki/pantheria
[35] https://eol.org/
[36] https://github.com/oeg-upm/valkyr-ie-gate

that prevents the generation of ad-hoc code that is not reusable to address novel challenges identified in new scenarios.

Future work of Helio will consist of adding new functionalities and improvements in the framework and providing tests for the framework. Regarding the improvement of the framework, the KG Curation Modules will be extended to add novel functionalities, such as ODRL policies [85]. In addition, the architecture of Helio will be broken down into pieces to constitute a distributed architecture capable of dealing with larger and more complex scenarios.

In the future, the Helio framework is going to be evaluated from different aspects. As it has been mentioned, the current Helio framework lacks formal experimentation, apart from a set of JUnit tests to check the correct software behaviour. In the future, the framework will be tested on different aspects. On the one hand, the RDF Generator Module is going to be tested with a benchmark from the literature, e.g., GTFS Bench [2]. The Publisher Module is going to be tested using a stress test tool like JMeter, obtaining time answering results for different operations, e.g., retrieving a resource or running a SPARQL query. Furthermore, the whole framework may be evaluated from a user perspective using questionnaires to get feedback about the usability of the whole framework.

On the other hand, it would be interesting to define a set of unitary tests for ensuring that a tool meets the requirements elicited in this article and that it conforms to the different standards that it uses.

8. Acknowledgements

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through the AURORAL project, Grant Agreement No. 101016854.

References

[1] M. Monti, F. Perego, Y. Zhao, G. Vetere, J.M. Gómez-Pérez, P. Alexopoulos, H.H. Nguyen, G. Webster, B. Villazón-Terrazas, N. García-Santa and J.Z. Pan, Success Stories, in: *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, J.Z. Pan, G. Vetere, J.M. Gómez-Pérez and H. Wu, eds, Springer, 2017, pp. 215–236. doi:10.1007/978-3-319-45654-6_8.

[2] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus and Ó. Corcho, GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain, *J. Web Semant.* **65** (2020), 100596. doi:10.1016/j.websem.2020.100596.

[3] D. Fensel, U. Simsek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich and A. Wahler, *Knowledge Graphs - Methodology, Tools and Selected Use Cases*, Springer, 2020. ISBN 978-3-030-37438-9. doi:10.1007/978-3-030-37439-6.

[4] C. Bizer, T. Heath and T. Berners-Lee, Linked Data - The Story So Far, *Int. J. Semantic Web Inf. Syst.* **5**(3) (2009), 1–22. doi:10.4018/jswis.2009081901.

[5] U. Simsek, J. Umbrich and D. Fensel, Towards a Knowledge Graph Lifecycle: A pipeline for the population of a commercial Knowledge Graph, in: *Proceedings of the Conference on Digital Curation Technologies (Qurator 2020), Berlin, Germany, January 20th - 21st, 2020*, A. Paschke, C. Neudecker, G. Rehm, J.A. Qundus and L. Pintscher, eds, CEUR Workshop Proceedings, Vol. 2535, CEUR-WS.org, 2020. http://ceur-ws.org/Vol-2535/paper_10.pdf.

[6] A. Dimou, M.V. Sande, P. Colpaert, R. Verborgh, E. Mannens and R.V. de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014*, C. Bizer, T. Heath, S. Auer and T. Berners-Lee, eds, CEUR Workshop Proceedings, Vol. 1184, CEUR-WS.org, 2014. http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[7] M. Nentwig, M. Hartung, A.N. Ngomo and E. Rahm, A survey of current Link Discovery frameworks, *Semantic Web* **8**(3) (2017), 419–436. doi:10.3233/SW-150210.

[8] J.M. Gómez-Pérez, J.Z. Pan, G. Vetere and H. Wu, Enterprise Knowledge Graph: An Introduction, in: *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, J.Z. Pan, G. Vetere, J.M. Gómez-Pérez and H. Wu, eds, Springer, 2017, pp. 1–14. doi:10.1007/978-3-319-45654-6_1.

[9] H. Weng, Z. Liu, S. Yan, M. Fan, A. Ou, D. Chen and T. Hao, A Framework for Automated Knowledge Graph Construction Towards Traditional Chinese Medicine, in: *Health Information Science - 6th International Conference, HIS 2017, Moscow, Russia, October 7-9, 2017, Proceedings*, S. Siuly, Z. Huang, U. Aickelin, R. Zhou, H. Wang, Y. Zhang and S.V. Klimenko, eds, Lecture Notes in Computer Science, Vol. 10594, Springer, 2017, pp. 170–181. doi:10.1007/978-3-319-69182-4_18.

[10] Q. Cong, Z. Feng, F. Li, L. Zhang, G. Rao and C. Tao, Constructing Biomedical Knowledge Graph Based on SemMedDB and Linked Open Data, in: *IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2018, Madrid, Spain, December 3-6, 2018*, H.J. Zheng, Z. Callejas, D. Griol, H. Wang, X. Hu, H.H.H.W. Schmidt, J. Baumbach, J. Dickerson and L. Zhang, eds, IEEE Computer Society, 2018, pp. 1628–1631. doi:10.1109/BIBM.2018.8621568.

[11] Y. Zhang, M. Sheng, R. Zhou, Y. Wang, G. Han, H. Zhang, C. Xing and J. Dong, HKGB: An Inclusive, Extensible, Intelligent, Semi-auto-constructed Knowledge Graph Framework for Healthcare with Clinicians' Expertise Incorporated, *Inf. Process. Manag.* **57**(6) (2020), 102324. doi:10.1016/j.ipm.2020.102324.

[12] T. Yu, J. Li, Q. Yu, Y. Tian, X. Shun, L. Xu, L. Zhu and H. Gao, Knowledge graph for TCM health preservation: Design, construction, and applications, *Artif. Intell. Medicine* **77** (2017), 48–52. doi:10.1016/j.artmed.2017.04.001.

[13] D. Fensel, U. Simsek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich and A. Wahler, *Knowledge Graphs*, Springer, 2020. ISBN 978-3-030-37438-9. doi:10.1007/978-3-030-37439-6.

[14] A. Cimmino, M. Poveda-Villalón and R. García-Castro, eWoT: A Semantic Interoperability Approach for Heterogeneous IoT Ecosystems Based on the Web of Things, *Sensors* **20**(3) (2020), 822. doi:10.3390/s20030822.

[15] J. Cano-Benito, A. Cimmino and R. García-Castro, Benchmarking the efficiency of RDF-based access for blockchain environments, in: *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, R. García-Castro, ed., KSI Research Inc., 2020, pp. 554–559. doi:10.18293/SEKE2020-104.

[16] A. Fernández-Izquierdo and R. García-Castro, Themis: a tool for validating ontologies through requirements, in: *The 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019*, A. Perkusich, ed., KSI Research Inc. and Knowledge Systems Institute Graduate School, 2019, pp. 573–753. doi:10.18293/SEKE2019-117.

[17] A. Cimmino, A. Fernández-Izquierdo and R. García-Castro, Astrea: Automatic Generation of SHACL Shapes from Ontologies, in: *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings*, A. Harth, S. Kirrane, A.N. Ngomo, H. Paulheim, A. Rula, A.L. Gentile, P. Haase and M. Cochez, eds, Lecture Notes in Computer Science, Vol. 12123, Springer, 2020, pp. 497–513. doi:10.1007/978-3-030-49461-2_29.

[18] A. Cimmino, V. Oravec, F. Serena, P. Kostelnik, M. Poveda-Villalón, A. Tryferidis, R. García-Castro, S. Vanya, D. Tzovaras and C. Grimm, VICINITY: IoT Semantic Interoperability Based on the Web of Things, in: *15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019*, IEEE, 2019, pp. 241–247. doi:10.1109/DCOSS.2019.00061.

[19] A. Cimmino, N. Andreadou, A. Fernández-Izquierdo, C. Patsonakis, A.C. Tsolakis, A. Lucas, D. Ioannidis, E. Kotsakis, D. Tzovaras and R. García-Castro, Semantic Interoperability for DR Schemes Employing the SGAM Framework, in: *2020 International Conference on Smart Energy Systems and Technologies (SEST)*, 2020, pp. 1–6. doi:10.1109/SEST48500.2020.9203338.

[20] J.C. de Benito, Oficina domótica semántica usando tecnología blockchain, 2019. http://oa.upm.es/55994/.

[21] L.G. Vélez, Sensorización de espacios de trabajo basado en el paradigma Web of Things, Master Thesis at Universidad Politécnica de Madrid, 2020. http://oa.upm.es/58136/.

[22] A.N. Molinero, Análisis de los factores implicados en el salto de huésped del virus de la rabia, Master Thesis at Universidad Politécnica de Madrid, 2019.

[23] D.N. Rodríguez, Espacios de trabajo inteligentes en la lucha contra la Covid-19, Master Thesis at Universidad Politécnica de Madrid, 2021. http://oa.upm.es/66302/.

[24] D. Chaves-Fraga, A. Alobaid, A. Cimmino, F. Priyatna and O. Corcho, Generating and querying (Virtual) Knowledge Graphs from heterogeneous data sources, in: *Tutorial at Extended Semantic Web Conference*, 2019.

[25] D. Chaves-Fraga, A. Iglesias-Molina, A.C. Arriaga and O. Corcho, Knowledge Graph Construction using Declarative Mapping Rules, in: *Tutorial at International Semantic Web Conference*, 2020.

[26] R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J.J. Carroll and B. McBride, RDF 1.1 concepts and abstract syntax, *W3C recommendation* **25**(02) (2014).

[27] P. Ristoski and H. Paulheim, RDF2Vec: RDF Graph Embeddings for Data Mining, in: *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, P. Groth, E. Simperl, A.J.G. Gray, M. Sabou, M. Krötzsch, F. Lécué, F. Flöck and Y. Gil, eds, Lecture Notes in Computer Science, Vol. 9981, 2016, pp. 498–514. doi:10.1007/978-3-319-46523-4_30.

[28] E. Amador-Domínguez, E. Serrano, D. Manrique, P. Hohenecker and T. Lukasiewicz, An ontology-based deep learning approach for triple classification with out-of-knowledge-base entities, *Information Sciences* **564** (2021), 85–102. doi:https://doi.org/10.1016/j.ins.2021.02.018. https://www.sciencedirect.com/science/article/pii/S0020025521001602.

[29] A. Cimmino and R. Corchuelo, On Feeding Business Systems with Linked Resources from the Web of Data, in: *Business Information Systems - 21st International Conference, BIS 2018, Berlin, Germany, July 18-20, 2018, Proceedings*, W. Abramowicz and A. Paschke, eds, Lecture Notes in Business Information Processing, Vol. 320, Springer, 2018, pp. 307–320. doi:10.1007/978-3-319-93931-5_22.

[30] H. Knublauch and D. Kontokostas, Shapes constraint language (SHACL), *W3C Candidate Recommendation* **11**(8) (2017).

[31] S. McCarron, B. Adida, M. Birbeck, G. Kellogg and I. Herman, HTML+ RDFa 1.1, *W3C recommendation* (2013).

[32] L. Feigenbaum, G.T. Williams, K.G. Clark and E. Torres, SPARQL 1.1 Protocol, *W3C Recommendation* (2013).

[33] Ó. Corcho, F. Priyatna and D. Chaves-Fraga, Towards a new generation of ontology based data access, *Semantic Web* **11**(1) (2020), 153–160. doi:10.3233/SW-190384.

[34] I. Herman, B. Adida, M. Sporny and M. Birbeck, RDFa 1.1 Primer—Third Edition, W3C Note (2015).

[35] U. Simsek, E. Kärle and D. Fensel, RocketRML - A NodeJS Implementation of a Use Case Specific RML Mapper, in: *Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics co-located with 16th Extended Semantic Web Conference (ESWC 2019), Portorož, Slovenia, June 3, 2019*, D. Chaves-Fraga, P. Heyvaert, F. Priyatna, J.F. Sequeda, A. Dimou, H. Jabeen, D. Graux, G. Sejdiu, M. Saleem and J. Lehmann, eds, CEUR Workshop Proceedings, Vol. 2489, CEUR-WS.org, 2019, pp. 46–53. http://ceur-ws.org/Vol-2489/paper5.pdf.

[36] S. Jozashoori and M. Vidal, MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation, in: *On the Move to Meaningful Internet Systems: OTM 2019 Conferences - Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21-25, 2019, Proceedings*, H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C.A. Ardagna and R. Meersman, eds, Lecture Notes

in Computer Science, Vol. 11877, Springer, 2019, pp. 58–75. doi:10.1007/978-3-030-33246-4_4.

[37] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh and A. Dimou, Parallel RDF generation from heterogeneous big data, in: *Proceedings of the International Workshop on Semantic Big Data, SBD@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*, S. Groppe and L. Gruenwald, eds, ACM, 2019, pp. 1:1–1:6. doi:10.1145/3323878.3325802.

[38] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL Extension for Generating RDF from Heterogeneous Formats, in: *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, eds, Lecture Notes in Computer Science, Vol. 10249, 2017, pp. 35–50. doi:10.1007/978-3-319-58068-5_3.

[39] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh and A. Dimou, Parallel RDF Generation from Heterogeneous Big Data, in: *Proceedings of the International Workshop on Semantic Big Data*, SBD '19, Association for Computing Machinery, New York, NY, USA, 2019. ISBN 9781450367660. doi:10.1145/3323878.3325802.

[40] A. Poggi, D. Lembo, D. Calvanese, G.D. Giacomo, M. Lenzerini and R. Rosati, Linking Data to Ontologies, 2008, pp. 133–173. doi:10.1007/978-3-540-77688-8_5.

[41] F. Priyatna, O. Corcho and J. Sequeda, Formalisation and Experiences of -based SPARQL to SQL Query Translation Using Morph, in: *International World Wide Web Conference*, ACM, New York, NY, USA, 2014, pp. 479–490. ISBN 978-1-4503-2744-2. doi:10.1145/2566486.2567981.

[42] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487. doi:10.3233/SW-160217.

[43] J.F. Sequeda and D.P. Miranker, Ultrawrap: SPARQL execution on relational data, *J. Web Semant.* **22** (2013), 19–39. doi:10.1016/j.websem.2013.08.002.

[44] F. Michel, L. Djimenou, C. Faron-Zucker and J. Montagnat, Translation of Relational and Non-relational Databases into RDF with xR2RML, in: *WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015*, V. Monfort, K. Krempels, T.A. Majchrzak and Z. Turk, eds, SciTePress, 2015, pp. 443–454. doi:10.5220/0005448304430454.

[45] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M.-E. Vidal and O. Corcho, Enhancing OBDA Query Translation over Tabular Data with MorphCSV, 2020.

[46] C. Bizer and A. Seaborne, D2RQ-treating non-RDF databases as virtual RDF graphs, in: *Proceedings of the 3rd international semantic web conference*, Vol. 2004, Proceedings of international semantic web conference, 2004.

[47] K.M. Endris, P.D. Rohde, M. Vidal and S. Auer, Ontario: Federated Query Processing Against a Semantic Data Lake **11706** (2019), 379–395. doi:10.1007/978-3-030-27615-7_29.

[48] M.N. Mami, D. Graux, S. Scerri, H. Jabeen and S. Auer, Querying Data Lakes using Spark and Presto, in: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, L. Liu, R.W. White, A. Mantrach, F. Silvestri, J.J. McAuley, R. Baeza-Yates and L. Zia, eds, ACM, 2019, pp. 3574–3578. doi:10.1145/3308558.3314132.

[49] Y. Khan, A. Zimmermann, A. Jha, V. Gadepally, M. d'Aquin and R. Sahay, One Size Does Not Fit All: Querying Web Polystores, *IEEE Access* **7** (2019), 9598–9617. doi:10.1109/ACCESS.2018.2888601.

[50] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, *Semantic Web* **8**(3) (2017), 489–508. doi:10.3233/SW-160218.

[51] D. Tomaszuk, RDF Validation: A Brief Survey, in: *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation - 13th International Conference, BDAS 2017, Ustroń, Poland, May 30 - June 2, 2017, Proceedings*, S. Kozielski, D. Mrozek, P. Kasprowski, B. Malysiak-Mrozek and D. Kostrzewa, eds, Communications in Computer and Information Science, Vol. 716, 2017, pp. 344–355. doi:10.1007/978-3-319-58274-0_28.

[52] X. Wang, L. Chen, T. Ban, M. Usman, Y. Guan, S. Liu, T. Wu and H. Chen, Knowledge Graph Quality Control: A Survey, *Fundamental Research* (2021). doi:https://doi.org/10.1016/j.fmre.2021.08.018. https://www.sciencedirect.com/science/article/pii/S2667325821001734.

[53] Z. Zheng, X. Luo and H. Wang, MESRG: multi-entity summarisation in RDF graph, *Int. J. Comput. Sci. Eng.* **23**(1) (2020), 74–81. doi:10.1504/IJCSE.2020.110197.

[54] A. Cimmino and R. Corchuelo, A Hybrid Genetic-Bootstrapping Approach to Link Resources in the Web of Data, in: *Hybrid Artificial Intelligent Systems - 13th International Conference, HAIS 2018, Oviedo, Spain, June 20-22, 2018, Proceedings*, F.J. de Cos Juez, J.R. Villar, E.A. de la Cal, Á. Herrero, H. Quintián, J.A. Sáez and E. Corchado, eds, Lecture Notes in Computer Science, Vol. 10870, Springer, 2018, pp. 145–157. doi:10.1007/978-3-319-92639-1_13.

[55] A.N. Ngomo and K. Lyko, EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming, in: *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, E. Simperl, P. Cimiano, A. Polleres, Ó. Corcho and V. Presutti, eds, Lecture Notes in Computer Science, Vol. 7295, Springer, 2012, pp. 149–163. doi:10.1007/978-3-642-30284-8_17.

[56] R. Isele and C. Bizer, Learning Expressive Linkage Rules using Genetic Programming, *Proc. VLDB Endow.* **5**(11) (2012), 1638–1649. doi:10.14778/2350229.2350276. http://vldb.org/pvldb/vol5/p1638_robertisele_vldb2012.pdf.

[57] R. Isele and C. Bizer, Active learning of expressive linkage rules using genetic programming, *J. Web Semant.* **23** (2013), 2–15. doi:10.1016/j.websem.2013.06.001.

[58] A. Nikolov, M. d'Aquin and E. Motta, Unsupervised Learning of Link Discovery Configuration, in: *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, E. Simperl, P. Cimiano, A. Polleres, Ó. Corcho and V. Presutti, eds, Lecture Notes in Computer Science, Vol. 7295, Springer, 2012, pp. 119–133. doi:10.1007/978-3-642-30284-8_15.

[59] T. Soru and A.N. Ngomo, A comparison of supervised learning classifiers for link discovery, in: *Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS 2014, Leipzig, Germany, September 4-5, 2014*, H. Sack, A. Filipowska, J. Lehmann and S. Hellmann, eds, ACM, 2014, pp. 41–44. doi:10.1145/2660517.2660532.

[60] A. Cimmino and R. Corchuelo, On Feeding Business Systems with Linked Resources from the Web of Data, in: *Business Information Systems - 21st International Conference, BIS 2018, Berlin, Germany, July 18-20, 2018, Proceedings*, W. Abramowicz and A. Paschke, eds, Lecture Notes in Business Information Processing, Vol. 320, Springer, 2018, pp. 307–320. doi:10.1007/978-3-319-93931-5_22.

[61] A. Cimmino, C.R. Rivero and D. Ruiz, Improving Link Specifications using Context-Aware Information, in: *Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, co-located with 25th International World Wide Web Conference (WWW 2016)*, S. Auer, T. Berners-Lee, C. Bizer and T. Heath, eds, CEUR Workshop Proceedings, Vol. 1593, CEUR-WS.org, 2016. http://ceur-ws.org/Vol-1593/article-08.pdf.

[62] A. Cimmino and R. Corchuelo, On learning context-aware rules to link RDF datasets, *Log. J. IGPL* **29**(2) (2021), 151–166. doi:10.1093/jigpal/jzaa043.

[63] I.F. Cruz, F.P. Antonelli and C. Stroe, AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies, *Proc. VLDB Endow.* **2**(2) (2009), 1586–1589. doi:10.14778/1687553.1687598. http://www.vldb.org/pvldb/vol2/vldb09-1003.pdf.

[64] J. Volz, C. Bizer, M. Gaedke and G. Kobilarov, Silk - A Link Discovery Framework for the Web of Data **538** (2009). http://ceur-ws.org/Vol-538/ldow2009_paper13.pdf.

[65] A.N. Ngomo and S. Auer, LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data, in: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, T. Walsh, ed., IJCAI/AAAI, 2011, pp. 2312–2317. doi:10.5591/978-1-57735-516-8/IJCAI11-385.

[66] D. Tomaszuk, RDF Validation: A Brief Survey, in: *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation - 13th International Conference, BDAS 2017, Ustroń, Poland, May 30 - June 2, 2017, Proceedings*, S. Kozielski, D. Mrozek, P. Kasprowski, B. Malysiak-Mrozek and D. Kostrzewa, eds, Communications in Computer and Information Science, Vol. 716, 2017, pp. 344–355. doi:10.1007/978-3-319-58274-0_28.

[67] S. Issa, O. Adekunle, F. Hamdi, S.S.-S. Cherfi, M. Dumontier and A. Zaveri, Knowledge Graph Completeness: A Systematic Literature Review, *IEEE Access* **9** (2021), 31322–31339. doi:10.1109/ACCESS.2021.3056622.

[68] H. Chen, G. Cao, J. Chen and J. Ding, A Practical Framework for Evaluating the Quality of Knowledge Graph, in: *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding - 4th China Conference, CCKS 2019, Hangzhou, China, August 24-27, 2019, Revised Selected Papers*, X. Zhu, B. Qin, X. Zhu, M. Liu and L. Qian, eds, Communications in Computer and Information Science, Vol. 1134, Springer, 2019, pp. 111–122. doi:10.1007/978-981-15-1956-7_10.

[69] B. Xue and L. Zou, Knowledge Graph Quality Management: a Comprehensive Survey, *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–1. doi:10.1109/TKDE.2022.3150080.

[70] B. McBride, Jena: a semantic Web toolkit, *IEEE Internet Computing* **6**(6) (2002), 55–59. doi:10.1109/MIC.2002.1067737.

[71] J. Broekstra, A. Kampman and F. van Harmelen, Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, in: *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, I. Horrocks and J.A. Hendler, eds, Lecture Notes in Computer Science, Vol. 2342, Springer, 2002, pp. 54–68. doi:10.1007/3-540-48005-6_7.

[72] R. Oldakowski, C. Bizer and D. Westphal, RAP: RDF API for PHP, in: *Proceedings of the Workshop on Scripting for the Semantic Web, SFSW 2005*, S. Auer, C. Bizer and L. Miller, eds, CEUR Workshop Proceedings, Vol. 135, CEUR-WS.org, 2005. https://ceur-ws.org/Vol-135/paper1.pdf.

[73] J. Wielemaker, G. Schreiber and B.J. Wielinga, Prolog-Based Infrastructure for RDF: Scalability and Performance, in: *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, D. Fensel, K.P. Sycara and J. Mylopoulos, eds, Lecture Notes in Computer Science, Vol. 2870, Springer, 2003, pp. 644–658. doi:10.1007/978-3-540-39718-2_41.

[74] E. Oren and R. Delbru, ActiveRDF: object-oriented RDF in Ruby, *Scripting for Semantic Web (ESWC)* (2006), 3.

[75] G.E. Modoni, M. Sacco and W. Terkaj, A survey of RDF store solutions, in: *2014 International Conference on Engineering, Technology and Innovation (ICE)*, 2014, pp. 1–7. doi:10.1109/ICE.2014.6871541.

[76] K. Rohloff, M. Dean, I. Emmons, D. Ryder and J. Sumner, An Evaluation of Triple-Store Technologies for Large Data Stores, in: *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, OTM Academy Doctoral Consortium, MONET, OnToContent, ORM, PerSys, PPN, RDDS, SSWS, and SWWS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II*, R. Meersman, Z. Tari and P. Herrero, eds, Lecture Notes in Computer Science, Vol. 4806, Springer, 2007, pp. 1105–1114. doi:10.1007/978-3-540-76890-6_38.

[77] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham and P. Castagna, Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store, in: *Proceedings of the 2012th International Conference on Posters & Demonstrations Track - Volume 914*, B. Glimm and D. Huynh, eds, CEUR Workshop Proceedings, Vol. 914, CEUR-WS.org, 2012. https://ceur-ws.org/Vol-914/paper_14.pdf.

[78] R. Punnoose, A. Crainiceanu and D. Rapp, Rya: A Scalable RDF Triple Store for the Clouds, in: *Proceedings of the 1st International Workshop on Cloud Intelligence*, Cloud-I '12, Association for Computing Machinery, New York, NY, USA, 2012. ISBN 9781450315968. doi:10.1145/2347673.2347677.

[79] M. Atre, J. Srinivasan and J.A. Hendler, BitMat: A main memory RDF triple store, *Tetherless World Constellation, Rensselar Plytehcnic Institute, Troy NY* (2009).

[80] L. Rietveld and R. Hoekstra, The YASGUI family of SPARQL clients, *Semantic Web* **8**(3) (2017), 373–383. doi:10.3233/SW-150197.

[81] R. Cyganiak and C. Bizer, Pubby-a linked data frontend for sparql endpoints, *http://wifo5-03.informatik.uni-mannheim.de/pubby/* (2008).

[82] A. Fernández-Izquierdo, A. Cimmino, C. Patsonakis, A.C. Tsolakis, R. García-Castro, D. Ioannidis and D. Tzovaras, OpenADR Ontology: Semantic Enrichment of Demand Response Strategies in Smart Grids, in: *2020 International Con-*

ference on Smart Energy Systems and Technologies (SEST), 2020, pp. 1–6. doi:10.1109/SEST48500.2020.9203093.

[83] A. Cimmino, M. McCool, F. Tavakolizadeh and K. Toumura, Web of Things (WoT) Discovery, *W3C Working Draft 2 June 2021* (2021).

[84] J. Cano-Benito, A. Cimmino and R. García-Castro, Towards Blockchain and Semantic Web, in: *Business Information Systems Workshops - BIS 2019 International Workshops, Seville, Spain, June 26-28, 2019, Revised Papers*, W. Abramowicz and R. Corchuelo, eds, Lecture Notes in Business Information Processing, Vol. 373, Springer, 2019, pp. 220–231. doi:10.1007/978-3-030-36691-9_19.

[85] R. Iannella M. and S. Villata, ODRL Information Model 2.2, *W3C Recommendation 15 February 2018* (2018).

[86] A. Fernández-Izquierdo, A. Cimmino and R. García-Castro, Supporting Demand-Response strategies with the DELTA ontology, in: *2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA)*, 2021, pp. 1–8. doi:10.1109/AICCSA53542.2021.9686935.

Appendix A. Json mapping serialisation for instantiating the Conceptual Mapping depicted by Figure 4

```
{
  "datasources" : [
      {
        "id" : "api_datasource",
        "refresh" : "3600",
        "type" : "JsonDatasource",
        "arguments" : ["$"],
        "connector"  : {
         "arguments" : ["http://sampledomain.com/api"],
          "type" : "URLConnector",
        }
      }
  ],
   "resource_rules" : [
    {
      "id" : "api_kg_data",
      "datasource_ids" : ["api_datasource"],
      "subject" : "http://kgdata.com/resource/sensor/{$.sensorId}",
      "properties"  : [
            {
               "predicate" : "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
               "object" : "https://w3id.org/saref#Measurement",
               "is_literal" : "False"
            },{
               "predicate" : "https://w3id.org/saref#relatesToProperty",
               "object" : "https://w3id.org/saref#Light",
               "is_literal" : "False"
            },{
               "predicate" : "https://w3id.org/saref#isMeasuredIn",
               "object" : "https://w3id.org/saref#IlluminanceUnit",
               "is_literal" : "False"
            },{
               "predicate" : "https://w3id.org/saref#hasValue",
               "object" : "{$.value}",
               "is_literal" : "True",
               "datatype" : "http://www.w3.org/2001/XMLSchema#float"
            },{
               "predicate" : "https://w3id.org/saref#hasTimestamp",
               "object" : "$.tmp",
               "is_literal" : "True",
               "datatype" : "http://www.w3.org/2001/XMLSchema#dateTime"
            }

      ]
    }
  ]
}
```

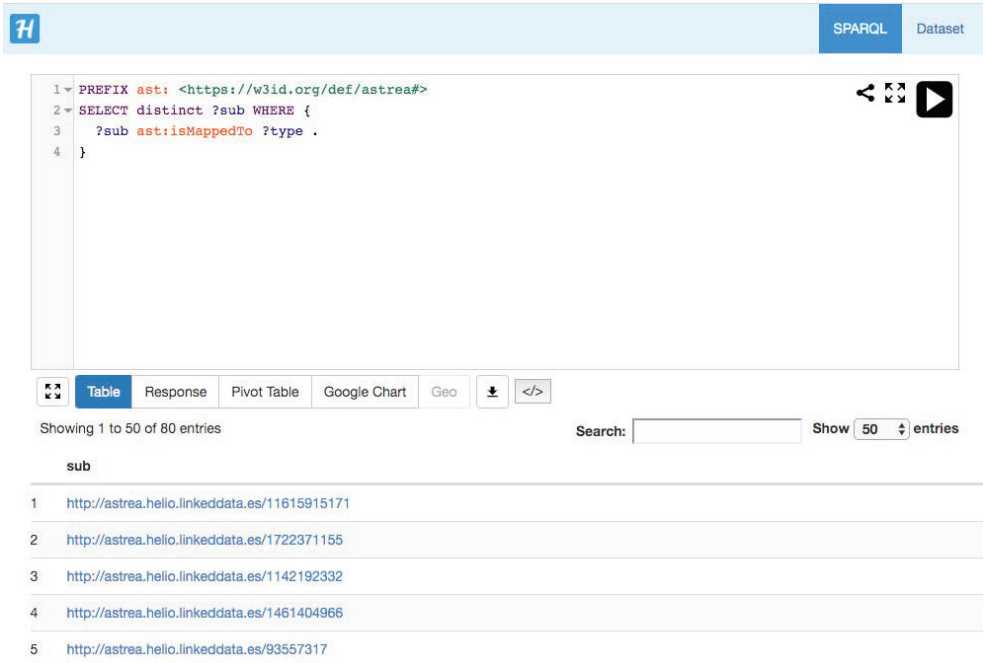
Appendix B. RML mapping serialisation for instantiating the Conceptual Mapping depicted by Figure 4

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
prefix saref: <https://w3id.org/saref#>

<#VenueMapping>
  rml:logicalSource [
    rml:source "http://sampledomain.com/";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$"
  ];

  rr:subjectMap [
    rr:template "http://kgdata.com/resource/sensor/{$.sensorId}";
    rr:class saref:Measurement
  ];

  rr:predicateObjectMap [
    rr:predicate saref:hasValue;
    rr:objectMap [
      rml:reference "$.value";
      rr:datatype xsd:float
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate saref:hasTimestamp;
    rr:objectMap [
      rml:reference "$.tmp";
      rr:datatype xsd:dateTime
    ]
  ];

  rr:predicateObjectMap [
    rr:predicate saref:isMeasuredIn;
    rr:object saref:IlluminanceUnit
  ];

  rr:predicateObjectMap [
    rr:predicate saref:relatesToProperty;
    rr:object saref:Light
  ].
```

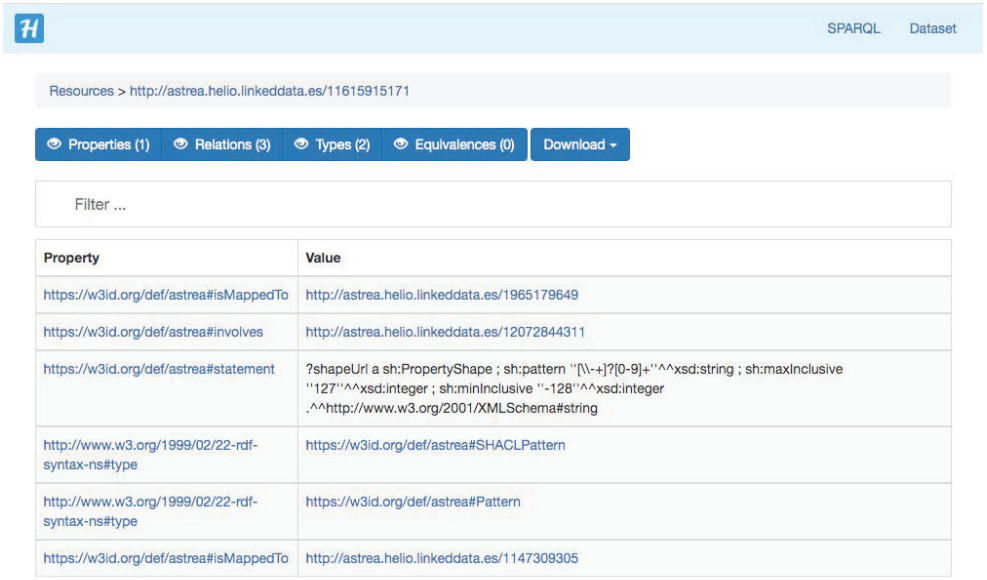
Appendix C. WoT-Mapping serialisation for instantiating the Conceptual Mapping depicted by Figure 4

```
@prefix wot: <http://iot.linkeddata.es/def/wot#> .
@prefix core: <http://iot.linkeddata.es/def/core#> .
@prefix map: <http://iot.linkeddata.es/def/wot-mappings#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix saref: <https://w3id.org/saref#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://kgdata.com/resource/sensor/001> core:isDescribedBy [
    a core:ThingDescription;
    core:describes <http://kgdata.com/resource/sensor/001>;
    map:hasAccessMapping [
      map:hasMapping [
        a map:ObjectProperty ;
        map:predicate rdf:type;
        map:targetClass saref:Measurement;
      ],[
        a map:ObjectProperty ;
        map:predicate saref:relatesToProperty;
        map:targetClass saref:Light;
      ],[
        a map:ObjectProperty ;
        map:predicate saref:isMeasuredIn;
        map:targetClass saref:IlluminanceUnit;
      ],[
        a map:ObjectProperty ;
        map:predicate saref:relatesToProperty;
        map:targetClass saref:Light;
      ],[
        a map:DataProperty ;
        map:predicate saref:hasValue;
        map:jsonPath "$.value";
        rdfs:Datatype xsd:float;
      ],[
        a map:DataProperty ;
        map:predicate saref:hasTimestamp;
        map:jsonPath "$.tmp";
        rdfs:Datatype xsd:dateTime;
      ];
      map:mapsResourceFrom [
        a wot:Link;
        wot:href "http://sampledomain.com/api/sensors?id=001";
        wot:mediaType "application/json" ;
      ]
    ]
  ].
```

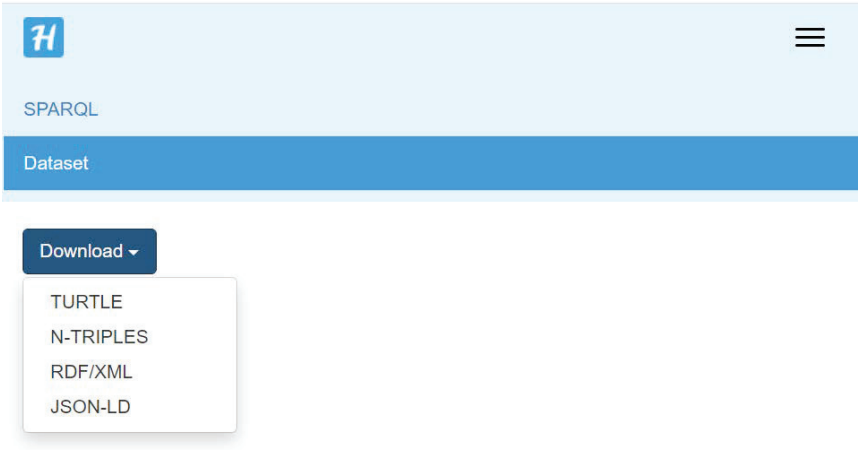
Appendix D. Helio Publisher Module HTML interfaces depicted by Figure 5



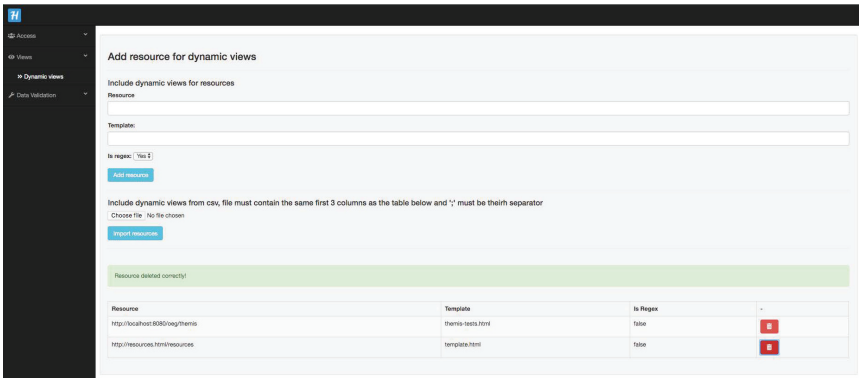
(a) SPARQL endpoint HTML interface.



(b) Default RDF HTML interface.



(c) Dataset HTML interface.



(d) Dynamic views and HTML customised menu

Figure 5. Helio Publisher Module interfaces.

Appendix E. Helio framework: Use Cases and real-world applications

The Helio framework provides to end-users several use cases related to the KG life cycle as depicted by Figure 6. Notice that the actor depicted is a user, which could be a person or a third-party application.

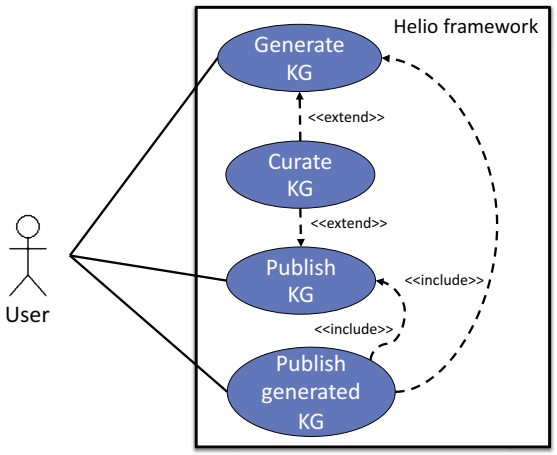


Figure 6. Use cases for KG life cycle

- Generate KG: the user generates the KG data from a set of heterogeneous sources (thanks to the KG Generator Module). As a result, the KG data is stored into a triple store (thanks to the Host Module).
- Publish KG: the user publishes a KG (thanks to the Publisher Module), with data that was previously stored in a triple store (thanks to the Host Module). Notice that this use case entails no generation of RDF data.
- Publish generated KG: the user builds the KG from a set of heterogeneous sources (thanks to the KG Generator Module). Then the KG data is stored in a triple store (thanks to the Host Module). Finally, the KG data is Published (thanks to the Publisher Module).
- Curate KG: this use case extends any of the other three aforementioned use cases. In any of them, this use case applies one or more curation techniques (thanks to the Curator Module).

In the light of the previous use cases, it can be inferred that some modules in the framework have a mandatory presence, whereas others are not mandatory. The modules that are mandatory are one of these combinations: KG Generator and the Host Module, Publisher and the Host Module, or KG Generator, Publisher, and Host Module. Besides, the Curator Module is optional in different use cases.

Notice that these use cases describe a high and abstract level of usage. Depending on a specific scenario, these use cases will be implemented differently. For instance, the use case *Generate KG* could be implemented as *Use the RDF Generator Module with RML mappings and store the result into GraphDB* or *Use the framework with a third party RDF generator storing the data into Fuseky*. Notice that this last implementation would require an end-user to code a *Provider* that wraps a third party RDF generator (like RMLMapper) and whose configuration points to the respective mappings. As a result, the *Provider* passes to the other components directly when the RDF is generated.

In the current Helio documentation, some of the implementations of the available use cases have been identified and documented[37]. Nevertheless, the authors would like to showcase how the framework was used in two different European projects:

Helio in DELTA: the goal of this project is providing a semantic interoperable platform for performing Demand Response among a wide number of infrastructures, including smart houses, residential buildings, or univer-

[37] https://github.com/oeg-upm/helio/wiki/Streamlined-use-cases

sity premises. In this platform, these infrastructures provide data about energy consumption and about other kinds of sensed data (like indoor temperature or humidity, weather records, etc.). In DELTA, these data are collected by edge nodes; then, at fog level, DELTA counts on several DELTA Virtual Nodes (DVN) that read these data and perform some Energy Demand Response over the edge nodes. Finally, at the cloud level, DELTA has a third component called Aggregator that reads data from the fog level and may also provide Demand Response signals to the DVNs, which would forward them to the edge level. The data exchanged in the DELTA platform follow the DELTA ontology [86].

The communication among these components is enabled thanks to the CIM gateway[38]. This artefact allows the edge nodes, the DVNs, and the Aggregator to communicate through an XMPP network. In addition, the CIM enables the consumption of data from the distributed components of DELTA using SPARQL Federation. Nevertheless, the main feature of the CIM is allowing edge nodes to translate the heterogeneous data coming from their infrastructures into RDF expressed according to the DELTA ontology.

The CIM relies on Helio for translating data from local infrastructures into a KG which is published on the XMPP cloud at two levels as RDF resources and through a SPARQL endpoint[39]. To abstract users from the development of mappings, and since the heterogeneity of the kind of data from the project is not wide, a GUI is developed to leverage the construction of mappings. This GUI allows users to define local REST endpoints and to associate interoperability modules to them[40]; i.e., mappings containing only translation rules from known payloads. As a result, users may in a user-friendly way configure the CIM for integrating their infrastructures (without realising that they are actually setting up the mappings).

In this project, Helio was used for translating heterogeneous data into RDF (namely, XML and JSON data). This data was stored into an embedded RDF4J triple store and published through XMPP, wrapping the regular Helio publisher API[41]. Furthermore, anytime an excerpt of RDF is generated, Helio also validates it using a SHACL Curator Module.

Helio in OntoCommons: this project is dedicated to the standardisation and harmonisation through ontologies of data documentation across all domains related to materials and manufacturing.

In this project, Helio is needed for publishing the wide number of ontologies gathered under the project scope (materials and manufacturing). The starting point of this scenario was an already curated KG containing these ontologies (rather than a set of heterogeneous data sources). As a result, Helio is used uniquely as a data publisher that provides a data portal with these ontologies within[42]. Notice that in this project, the features of Helio for customising the HTML views are used for providing appealing interfaces for interacting with this data portal.

[38] https://github.com/oeg-upm/DeltaCimApp
[39] https://www.delta-h2020.eu/wp-content/uploads/2019/11/DELTA_D3.1_final.pdf
[40] https://www.delta-h2020.eu/wp-content/uploads/2021/01/DELTA_D3.2_Final.pdf
[41] https://github.com/oeg-upm/helio/wiki/Helio-Publisher#linked-data-service
[42] https://data.ontocommons.linkeddata.es/index