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: {cimmino, rgarcia}@fi.upm.es

Abstract. Building and publishing knowledge graphs (KG) as Linked Data, either in 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 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 corpora 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 which are not generic, and thus, non re-usable. 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 at covering a set of requirements elicited from the KG life cycle and providing a tool capable of performing the different tasks required to build and publish KGs. As a result, Helio reduces the effort required to perform this process and prevents the development of ad-hoc pipelines. The Helio framework has been applied in a wide number of contexts, from European projects to research work.

Keywords: Knowledge graph generation, Knowledge graph publication, Link Data

1. Introduction

The presence of knowledge graphs (KGs) published openly on the Web, or privately as Linked Data has growth in the last decade [1]. The reason of 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 has to be built using Semantic Web Technologies [3] like RDF and, then, published following the Linked Data principles [4]. In fact, 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 by means of link discovery tools [7].

There is a large number of tools that 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 at solving a reduced set of very specific tasks; usually involved with 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 automatically is not possible without developing custom ad-hoc code in most of the cases [5]. Up to the authors' knowledge there is no tool that is able to cope and cover all the tasks conforming a KG life cycle which are required for building and publishing KGs [5].

As a consequence, 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 from which some are research prototypes that are not suitable for a production environment or they lack of fundamental documentation hindering their usability. 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 pipe lines to build and publish a KG [9–12]. Developing these ad-hoc pipelines has a high cost in time, personal resources, and requires large cycles of debugging and maintenance, decreasing the productivity of a project.

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 integrating these without modifying the framework source code by means of 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, also, 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 of the existing tools to perform these tasks and, on the other hand, allows practitioners to extend the framework in order to cope with new scenarios. Helio fosters the development of plugins for either using existing tools or implementing new functionalities since they are highly reusable. As a result, the use of plugins prevents developing ad-hoc pipelines, and allows 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 relayed 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 a 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.

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 task are usually related to one or more research problems, which are still active nowadays or they lack of a recommendation; fostering the numerous existing tools that tackle the same problem. As Figure 1 depicts, some of these task are also related to the Linked Data principles. The different steps of the life cycle are the following:

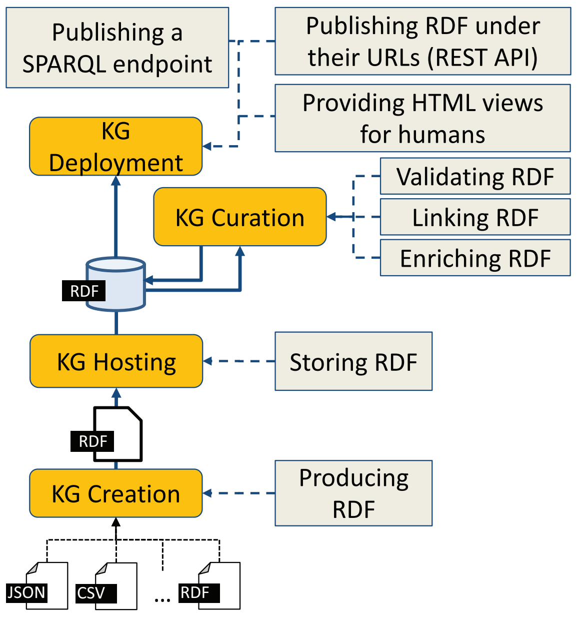


Figure 1. KG life cycle [13] and related tasks.

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

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 the data from heterogeneous data sources by means of translation mappings [6]. practitioners rely on materialisers when the KG data must provide an 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 automatically the data 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 create 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 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 an HTML document (lacking its formal semantics); instead, for machines the RDF data is published at resource level by means of a URL in a REST API that provides the RDF of such resource. A mixed solution proposed by the W3C consists of using HTML+RDFa documents [31], publishing HTML documents understandable by humans that contain RDF annotations understandable for 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 which 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 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 needing to learn a new mapping language.
- (KG Creation) R07: The system provides reusable extension mechanisms that allow to extend 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 the RDF data for a 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 Curation) R11: The system provides at least one technique for cleaning, enriching, validating, or linking RDF data that has been previously stored by the system.
- (KG Deployment) R12: 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) R13: 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) R14: The system provides content-negotiation so practitioners can consume the RDF data and/or the SPARQL results in different serialisations.
- (KG Deployment) R15: The system publishes HTML views for assisting practitioners during the data consumption.
- (KG Deployment) R16: The system provides mechanisms to customise the HTML views, e.g., to allow practitioners to change the aesthetics of the HTML views.
- (KG Deployment) R17: 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 R12; 3) When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL), covered by R12 and R13; and 4) Include links to other URIs, so that they can discover more things, covered by R10 and/or R11.

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 ($\sim$). 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 implementing the other KG creation requirements: only 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

		Categories of tools from the literature				
KG life cycle	Requirements	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	-	-	∼	-	-
	R11	-	-	∼	-	-
KG Deployment	R12	-	-	∼	∼	-
	R13	-	∼	∼	✓	∼
	R14	-	-	∼	✓	∼
	R15	-	-	-	-	∼
	R16	-	-	-	-	∼
	R17	-	-	-	-	-

Table 1

Elicited requirements meet by existing tools types

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 are 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.

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 translate 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 form 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 (R13), 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 R13 for some tools.

3.2. Knowledge graph curation

KG curation involves a large number of tools from the literature that can be divided into three categories: RDF enriching tools [50], RDF link discovery tools [7], and RDF validation tools [51].

Notice that the KG curation is a step that occurs once the RDF data has been stored after the KG Hosting. It is worthwhile to mention that the tasks involved in KG curation are not 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 and R11; however, by themselves they do not cover any requirement from 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 [52].

RDF link discovery aims at producing relationships among 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 [53–61], i.e., restrictions under which two RDF resources are linked. On the other hand, other tools focus on applying efficiently those rules and producing the links among resources [62–64].

RDF validators are tools that specify whether data expressed in RDF conforms with a set of restrictions. There is a specification for expressing these restrictions that is a W3C standard, i.e., SHACL [30], and other non-standard specifications [65]. These tools usually take as input an excerpt of RDF data and a set of restrictions, and produce a validation report.

3.3. Knowledge graph hosting and deployment

RDF frameworks aim at providing practitioners several functionalities, e.g., pragmatically chose different environments where to host their RDF data, that belong to different steps of the KG life cycle [66–70]. Some tools like Star Dog[5] 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 [66], 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).

Despite the RDF frameworks, in most of the cases the KGs are deployed by storing their RDF data into triple stores [71–75]. These stores host the RDF data and also provide a SPARQL endpoint (R13). Some triple stores also publish each resource under a URL (R12) and others implement content-negotiation for the SPARQL endpoint and the RDF resources (R14), including HTML documents. Nevertheless, triple stores are not suitable for any other task within the KG life cycle.

RDF publishers aim at providing human interfaces (HTML) for those SPARQL endpoints and resources published exclusively with machine interfaces. Some tools, like YASGUI [76], publish a human SPARQL interface for a given SPARQL endpoint (R13 and R14). Others, like Pubby [77], also publish human interfaces for the resources provided by the SPARQL endpoint (R15). Some others, like Elda[7], allow to customise the aesthetics of HTML views meant for humans relying on templates (R16). Nevertheless, up to the authors' knowledge, none of these tools allows the customisation of HTML views for transforming them into HTML+RDFa [34] (R17).

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 publishing 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 a Java artefacts although they could be implemented, or viewed, as micro-services alternatively. The modules and the requirements that they cover, depicted in Figure 2, are the following:

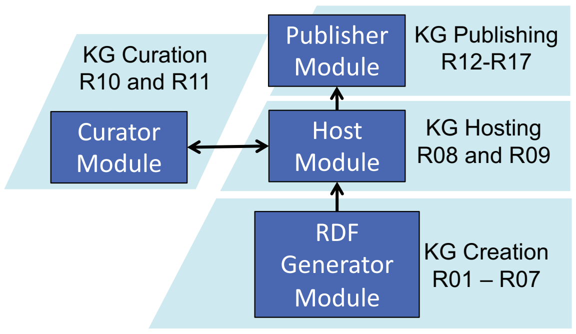


Figure 2. Helio framework

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 of RDF data that are passed to the Hosting Module. This module aims at covering all the requirements related to

[5]https://www.stardog.com/
[6]https://rdf4j.org/
[7]http://epimorphics.github.io/elda/index.html
[8]https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users

the 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 an 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 stored data. This module aims at covering all 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 all the requirements related to the KG Curation, namely R10 and R11. 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 all the requirements related to the KG Publishing, namely from R12 to R17. Section 4.4 provides a detailed description of this module.

In the following sub-sections 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.

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 a Helio bespoke mapping language, the conceptual mapping, that must be provided to the RDF Generator Module as input. Additionally, there is a last component named resources orchestrator that organises the whole translation process and also pushes the generated RDF data into the Hosting Module.

The data providers are components in charge of retrieving the 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 from the protocols involved for retrieved such data. The current Helio implementation counts with several data handlers instantiations[12]. *Example 2: assuming that the data retrieved in the 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 expression, 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,

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

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

[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

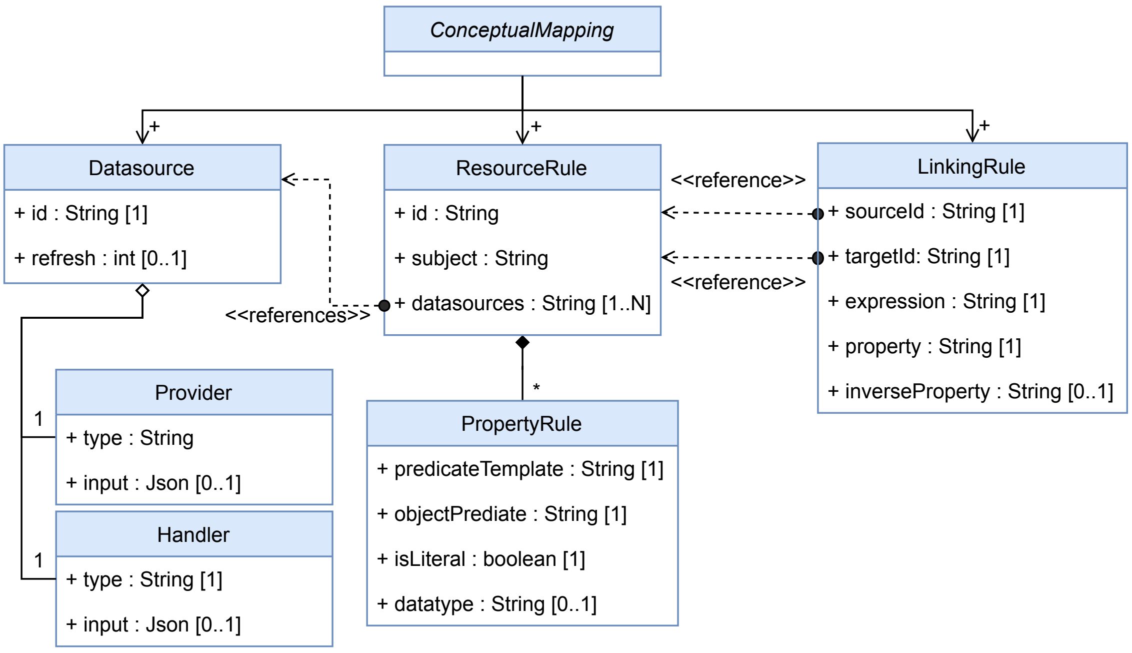


Figure 3. Helio Conceptual Mappings model

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 stand by. In case that 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 result.

The resources orchestrator is the component that triggers the RDF translator when required on-demand by any other module. In 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 that on-demand.

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.

A *ResourceRule* describes how the data from one or more sources is translated into RDF. It has a uniquely 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*

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

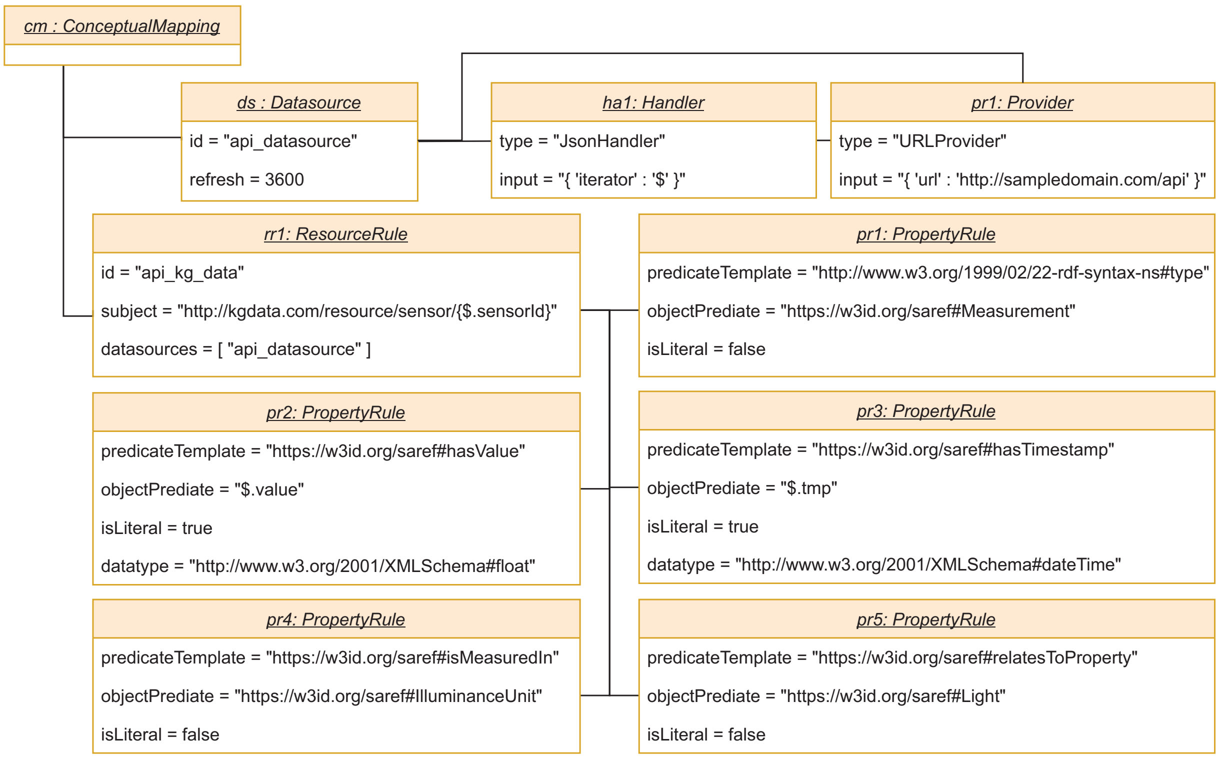

cm : ConceptualMapping
ds : Datasource
id = "api_datasource"
refresh = 3600
ha1: Handler
type = "JsonHandler"
input = "{ 'iterator' : '$' }"
pr1: Provider
type = "URLProvider"
input = "{ 'url' : 'http://sampledomain.com/api' }"
rr1: ResourceRule
id = "api_kg_data"
subject = "http://kgdata.com/resource/sensor/{$.sensorId}"
datasources = ["api_datasource"]
pr1: PropertyRule
predicateTemplate = "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
objectPrediate = "https://w3id.org/saref#Measurement"
isLiteral = false
pr2: PropertyRule
predicateTemplate = "https://w3id.org/saref#hasValue"
objectPrediate = "$.value"
isLiteral = true
datatype = "http://www.w3.org/2001/XMLSchema#float"
pr3: PropertyRule
predicateTemplate = "https://w3id.org/saref#hasTimestamp"
objectPrediate = "$.tmp"
isLiteral = true
datatype = "http://www.w3.org/2001/XMLSchema#dateTime"
pr4: PropertyRule
predicateTemplate = "https://w3id.org/saref#isMeasuredIn"
objectPrediate = "https://w3id.org/saref#IlluminanceUnit"
isLiteral = false
pr5: PropertyRule
predicateTemplate = "https://w3id.org/saref#relatesToProperty"
objectPrediate = "https://w3id.org/saref#Light"
isLiteral = false

Figure 4. Example of Conceptual Mappings for a REST API

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 [60], 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).

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 handle internally. This entails that 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 also linking the RDF resources generated. Although it counts with several data providers and handlers for achieving this task, new scenarios may introduce protocols or formats currently not supported by the module.

[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

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 this 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 to mention that, similarly to materialisers, OBDI or OBDA techniques could be also 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 the 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 the 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 [
      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.

[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

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 who publishes the SPARQL endpoint, perform the desired curation task, and then store the output by using the Hosting Module though 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 re-used by any third-party entity that will have to deal with a similar, or the same, curation challenge.

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 all 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 [76]) (shown by Figure 5a), any RDF resource (shown by Figure 5b), and the dataset (shown by Figure 5c).

Besides the standard views for 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 to associate 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 on 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.

5. Discussion

This section aims at providing a discussion divided into two 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.

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 all 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 splitted into the same steps, easing the coverage analysis of these requirements.

The Generator Module meets all 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 defining cleaning functions (**R04**) or linking rules (**R05**). Finally, the Generator Module implements a plugin-based system that prevents users to develop ad-hoc code that is not reusable (**R07**). Relying on this mechanism 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 the conceptual mappings for pushing the data synchronously on-demand or asynchronously periodically each quantum of time.

The Curation Module allows, thanks to the architecture of the framework, to plug any existing tool based on SPARQL in order to clean, enrich, validate, or link (**R10**). Nevertheless, the Generator Module also allows to define cleaning functions and link rules for resources in the dataset (**R11**).

Finally, the Publisher Module meets all the requirements related to the KG Deployment, i.e., **R12-R17**. The publisher provides a REST API for accessing the RDF resources (**R12**), perform queries using a standard SPARQL endpoint (**R13**), and download the KG dump. Furthermore, relaying on HTTP content negotiation the RDF resources, or the dump, are provided in different formats (**R14**). For HTML MIME types the Publisher Module already provides a set of default views (**R15**). Nevertheless, it also allows defining custom HTML views (**R16**) that may include meta-annotations transforming HTML into HTML+RDFa (**R17**).

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.

6. Helio framework adoption

The Helio framework lacks of a 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[22]: 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], which ontology is publicly available[23]. In the context of this project this component was the heart of the semantic interoperability approach responsible to allow components to transparently exchange and understand data. In this context, Helio was extended with the WoT-Mappings translator.

[22]https://www.vicinity2020.eu/vicinity/

[23]http://iot.linkeddata.es/def/wot-mappings

As 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[24]**:** in the European project DELTA Helio was used to implement the whole semantic interoperability architecture [19]. In this project, Helio was part of the Common Information Model (CIM)[25] software that enables components to exchange data in different formats. Then the CIM offered the services to exchange data (by translating the heterogeneous data on the fly into RDF expressed according to the DELTA ontology [78]), 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 templates for publishing the data and also enable a validation mechanism to ensure data quality.

AURORAL[26] 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 [79] 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[27]**:** 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.

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 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[28].

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[29].

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, 80] 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 blockchains regardless 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 a control service 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 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[30], some data from the Encyclopedia of Life[31], and Wikidata.

NLP: in the context of the Natural Language Processor, Helio is been currently used to assist Name Entity Recognition (NER) tasks. For this purpose the Valkyr-IE plugin has been developed, which allows to extract entities from a set of given texts using the tool Valkyr-IE[32].

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

[24]https://www.delta-h2020.eu/
[25]https://github.com/oeg-upm/DeltaCimApp
[26]https://www.auroral.eu
[27]https://cogito-project.eu/
[28]https://astrea.helio.linkeddata.es/
[29]http://themis.linkeddata.es/catalogue.html
[30]https://ecologicaldata.org/wiki/pantheria
[31]https://eol.org/
[32]https://github.com/oeg-upm/valkyr-ie-gate

7. Conclusions

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-cicle of the KGs, meeting these requirements and also allowing practitioners to publish KGs following the Linked Data principles. Furthermore, the framework counts with a plugin system that prevents the generation of ad-hoc code that is not reusable to address novel challenges identified in new scenarios.

Although the presented framework lacks of a formal experimentation, its wide adoption presents an indicator of its usability and usefulness. Future work in Helio will follow two paths: on the one hand, the KG curation modules available for Helio will be extended to add novel functionalities, such as ODRL policies [81]; on the other hand, the architecture of Helio will be break-down into pieces to constitute a distributed architecture capable of dealing with larger and complex scenarios.

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. Gomez-Perez, P. Alexopoulos, H. Nguyen, G. Webster, B. Villazon-Terrazas, N. Garcia-Santa and J.Z. Pan, *Success Stories*, in: *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, J.Z. Pan, G. Vetere, J.M. Gomez-Perez and H. Wu, eds, Springer International Publishing, Cham, 2017, pp. 215–236. ISBN 978-3-319-45654-6. doi:10.1007/978-3-319-45654-6_8. https://doi.org/10.1007/978-3-319-45654-6_8.

[2] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus and O. Corcho, GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain, *Journal of Web Semantics* **65** (2020), 100596.

[3] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich and A. Wahler, *How to Build a Knowledge Graph*, in: *Knowledge Graphs: Methodology, Tools and Selected Use Cases*, Springer International Publishing, Cham, 2020, pp. 11–68. ISBN 978-3-030-37439-6. doi:10.1007/978-3-030-37439-6_2.

[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.-C. Ngonga Ngomo and E. Rahm, A survey of current link discovery frameworks, *Semantic Web* **8**(3) (2017), 419–436.

[8] J.M. Gomez-Perez, J.Z. Pan, G. Vetere and H. Wu, Enterprise knowledge graph: An introduction, in: *Exploiting linked data and knowledge graphs in large organisations*, Springer, 2017, pp. 1–14.

[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*, S. Siuly, Z. Huang, U. Aickelin, R. Zhou, H. Wang, Y. Zhang and S. Klimenko, eds, Springer International Publishing, Cham, 2017, pp. 170–181. ISBN 978-3-319-69182-4.

[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: *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 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, *Information Processing & Management* **57**(6) (2020), 102324. doi:https://doi.org/10.1016/j.ipm.2020.102324. http://www.sciencedirect.com/science/article/pii/S0306457320308190.

[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, *Artificial Intelligence in Medicine* **77** (2017), 48–52. doi:https://doi.org/10.1016/j.artmed.2017.04.001. http://www.sciencedirect.com/science/article/pii/S0933365717301355.

[13] 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.

[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.

[15] A. Cimmino, R. García-Castro and J. Cano-Benito, 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. Garcıa-Castro, Themis: a tool for validating ontologies through requirements, in: *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 2019*, KSI Research Inc. and Knowledge Systems I, 2019, pp. 573–578.

[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: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, IEEE, 2019, pp. 241–247.

[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)*, IEEE, 2020, pp. 1–6.

[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. http://oa.upm.es/57068/.

[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. https://tutorials.oeg-upm.net/vkg2019/.

[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. https://tutorials.oeg-upm.net/kgc2020/.

[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*, P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck and Y. Gil, eds, Springer International Publishing, Cham, 2016, pp. 498–514.

[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, 2013.

[32] L. Feigenbaum, G.T. Williams, K.G. Clark and E. Torres, SPARQL 1.1 Protocol, *Recommendation, W3C, March* (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, *arXiv preprint arXiv:1903.04969* (2019).

[36] S. Jozashoori and M.-E. Vidal, MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation, in: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2019, pp. 58–75.

[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*, 2019, pp. 1–6.

[38] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: *European Semantic Web Conference*, Springer, 2017, pp. 35–50.

[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. De Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, in: *Journal on Data Semantics X*, Springer, 2008, pp. 133–173.

[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.

[43] J.F. Sequeda and D.P. Miranker, Ultrawrap: SPARQL execution on relational data, *Web Semantics: Science, Services and Agents on the WWW* **22** (2013), 19–39. doi:https://doi.org/10.1016/j.websem.2013.08.002. http://www.sciencedirect.com/science/article/pii/S1570826813000383.

[44] F. Michel, L. Djimenou, C.F. Zucker and J. Montagnat, Translation of relational and non-relational databases into RDF with xR2RML, in: *11th International Confenrence on Web Information Systems and Technologies (WEBIST'15)*, 2015, pp. 443–454.

[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 (ISWC2004)*, Vol. 2004, Proceedings of ISWC2004, 2004.

[47] K.M. Endris, P.D. Rohde, M.-E. Vidal and S. Auer, Ontario: Federated Query Processing against a Semantic Data Lake, *Database and Expert Systems Applications. Lecture Notes in Computer Science. Springer, Cham* (2019).

[48] M.N. Mami, D. Graux, S. Scerri, H. Jabeen and S. Auer, Querying Data Lakes using Spark and Presto, in: *International World Wide Web Conference*, ACM, 2019, pp. 3574–3578.

[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.

[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: *International Conference: Beyond Databases, Architectures and Structures*, Springer, 2017, pp. 344–355.

[52] Z. Zheng, X. Luo and H. Wang, MESRG: multi-entity summarisation in RDF graph, *International Journal of Computational Science and Engineering* **23**(1) (2020), 74–81.

[53] A. Cimmino and R. Corchuelo, A hybrid genetic-bootstrapping approach to link resources in the web of data, in: *International Conference on Hybrid Artificial Intelligence Systems*, Springer, 2018, pp. 145–157.

[54] A.-C.N. Ngomo and K. Lyko, EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming, in: *ESWC*, 2012, pp. 149–163.

[55] R. Isele and C. Bizer, Learning Expressive Linkage Rules using Genetic Programming, *PVLDB* **5**(11) (2012), 1638–1649.

[56] R. Isele and C. Bizer, Active learning of expressive linkage rules using genetic programming, *J. Web Sem.* **23** (2013), 2–15.

[57] A. Nikolov, M. d'Aquin and E. Motta, Unsupervised Learning of Link Discovery Configuration, in: *ESWC*, 2012, pp. 119–133.

[58] T. Soru and A.-C.N. Ngomo, A comparison of supervised learning classifiers for link discovery, in: *SEMANTICS*, 2014, pp. 41–44.

[59] A. Cimmino and R. Corchuelo, On Feeding Business Systems with Linked Resources from the Web of Data, in: *BIS*, 2018, pp. 307–320.

[60] A. Cimmino, C.R. Rivero and D. Ruiz, Improving Link Specifications using Context-Aware Information., in: *LDOW@ WWW*, 2016.

[61] A. Cimmino and R. Corchuelo, On learning context-aware rules to link RDF datasets, *Logic Journal of the IGPL* (2020).

[62] I.F. Cruz, F.P. Antonelli and C. Stroe, AgreementMaker: efficient matching for large real-world schemas and ontologies, *Proceedings of the VLDB Endowment* **2**(2) (2009), 1586–1589.

[63] J. Volz, C. Bizer, M. Gaedke and G. Kobilarov, Silk-a link discovery framework for the web of data., *Ldow* **538** (2009), 53.

[64] A.-C.N. Ngomo and S. Auer, LIMES—a time-efficient approach for large-scale link discovery on the web of data, in: *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[65] 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.

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

[67] J. Broekstra, A. Kampman and F. Van Harmelen, Sesame: A generic architecture for storing and querying rdf and rdf schema, in: *International semantic web conference*, Springer, 2002, pp. 54–68.

[68] R. Oldakowski, C. Bizer and D. Westphal, Rap: Rdf api for php, in: *Proceedings of the Workshop Scripting for the Semantic Web*, 2005.

[69] J. Wielemaker, G. Schreiber and B. Wielinga, Prolog-Based Infrastructure for RDF: Scalability and Performance, in: *The Semantic Web - ISWC 2003*, D. Fensel, K. Sycara and J. Mylopoulos, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 644–658. ISBN 978-3-540-39718-2.

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

[71] 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.

[72] 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*, R. Meersman, Z. Tari and P. Herrero, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 1105–1114. ISBN 978-3-540-76890-6.

[73] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham and P. Castagna, Jena-HBase: A distributed, scalable and efficient RDF triple store, in: *Proceedings of the 11th International Semantic Web Conference Posters and Demonstrations Track, ISWC-PD*, Vol. 12, Citeseer, 2012, pp. 85–88.

[74] 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.

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

[76] L. Rietveld and R. Hoekstra, The YASGUI family of SPARQL clients 1, *Semantic Web* **8**(3) (2017), 373–383.

[77] R. Cyganiak and C. Bizer, Pubby-a linked data frontend for sparql endpoints, *Url: http://wifo5-03. informatik. uni-mannheim. de/pubby/.(Acesso: 19-11-2014)* (2008).

[78] 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 Conference on Smart Energy Systems and Technologies (SEST)*, IEEE, 2020, pp. 1–6.

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

[80] J. Cano-Benito, A. Cimmino and R. García-Castro, Towards Blockchain and Semantic Web, in: *International Conference on Business Information Systems*, Springer, 2019, pp. 220–231.

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

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"
            }

      ]
    }
  ]
}
```

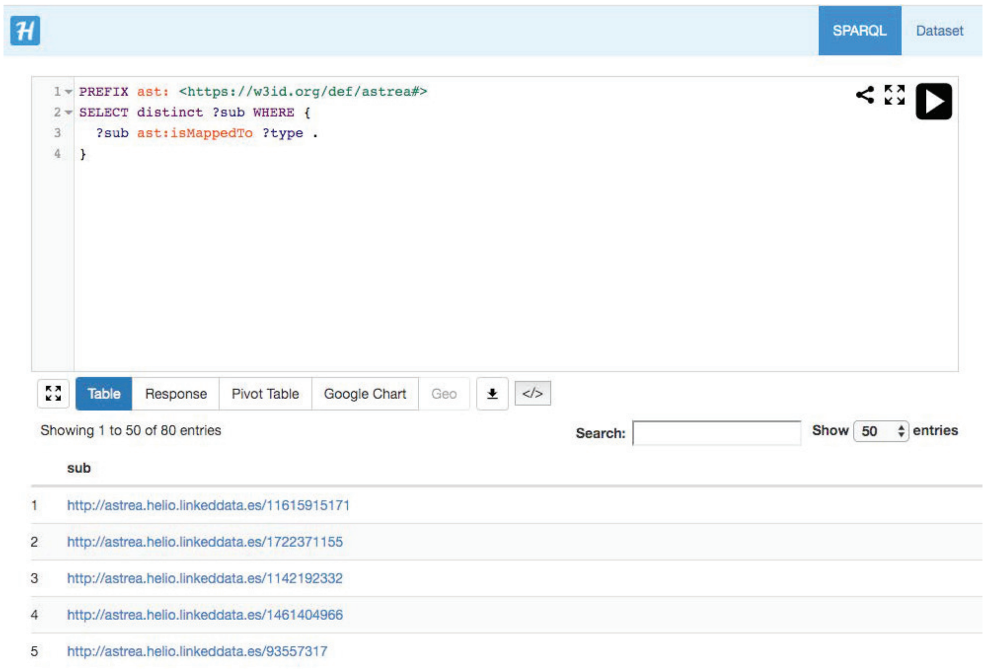
Appendix B. RML mapping serialisation for instantiating the Conceptual Mapping depicted by Figure 4

```
@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

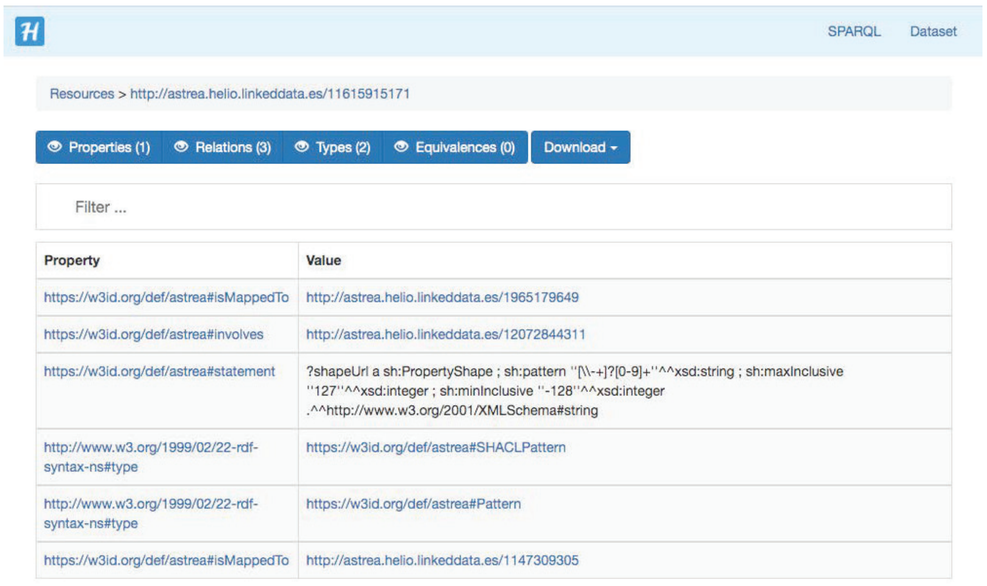


```
PREFIX ast: <https://w3id.org/def/astrea#>
SELECT distinct ?sub WHERE {
  ?sub ast:isMappedTo ?type .
}
```

Showing 1 to 50 of 80 entries

	sub
1	http://astrea.helio.linkeddata.es/11615915171
2	http://astrea.helio.linkeddata.es/1722371155
3	http://astrea.helio.linkeddata.es/1142192332
4	http://astrea.helio.linkeddata.es/1461404966
5	http://astrea.helio.linkeddata.es/93557317

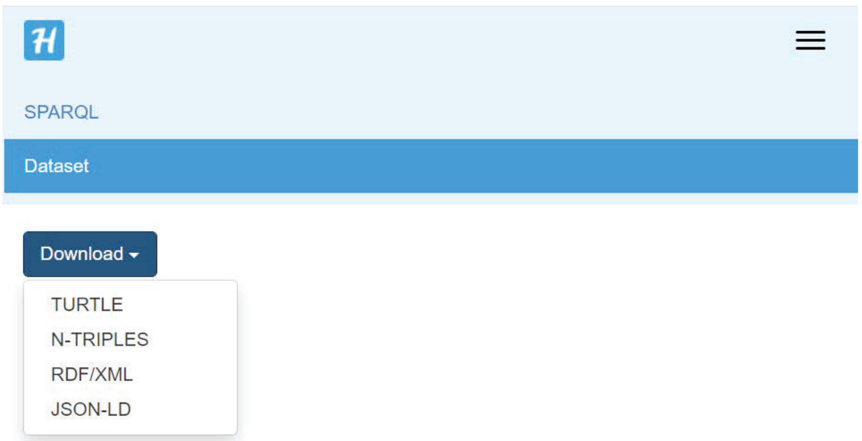
(a) SPARQL endpoint HTML interface



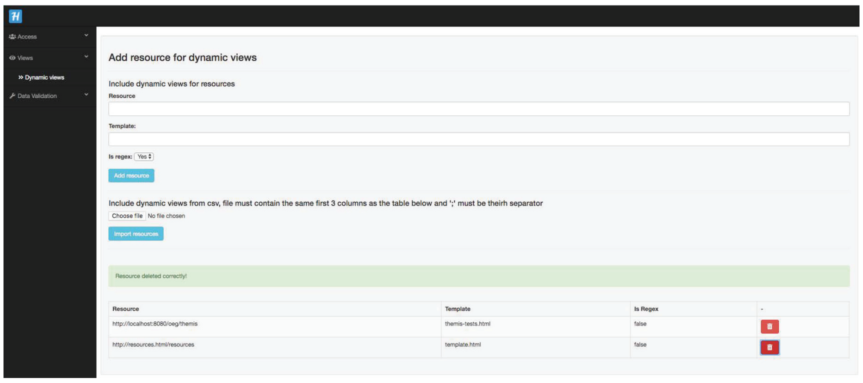
Resources > http://astrea.helio.linkeddata.es/11615915171

Property	Value
https://w3id.org/def/astrea#isMappedTo	http://astrea.helio.linkeddata.es/1965179649
https://w3id.org/def/astrea#involves	http://astrea.helio.linkeddata.es/12072844311
https://w3id.org/def/astrea#statement	?shapeUrl a sh:PropertyShape ; sh:pattern ''[\\-+]?[0-9]+''^^xsd:string ; sh:maxInclusive ''127''^^xsd:integer ; sh:minInclusive ''-128''^^xsd:integer .^^http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://w3id.org/def/astrea#SHACLPattern
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	https://w3id.org/def/astrea#Pattern
https://w3id.org/def/astrea#isMappedTo	http://astrea.helio.linkeddata.es/1147309305

(b) Default RDF HTML interface



(c) Dataset HTML interface



Add resource for dynamic views

Include dynamic views for resources

Resource

Template:

Is regex: Yes

Add resource

Include dynamic views from csv, file must contain the same first 3 columns as the table below and ';' must be theirh separator

Choose file No file chosen

Import resources

Resource deleted correctly!

Resource	Template	Is Regex	-
http://localhost:8080/oeg/themis	themis-tests.html	false	🟥
http://resources.html/resources	template.html	false	🟥

(d) Dynamic views and HTML customised menu

Figure 5. Helio Publisher Module interfaces