An Ontological Approach for Representing Declarative Mapping Languages

Ana Iglesias-Molina [a,*], Andrea Cimmino [a], Edna Ruckhaus [a], David Chaves-Fraga [a,b,c], Raúl García-Castro [a] and Oscar Corcho [a]

[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
E-mails: ana.iglesiasm@upm.es, andreajesus.cimmino@upm.es, e.ruckhaus@upm.es, david.chaves@upm.es, r.garcia@upm.es, oscar.corcho@upm.es
[b] *Declarative Languages and Artificial Intelligence Group, KU Leuven, Belgium*
[c] *Flanders Make, DTAI-FET, Belgium*

Editor: Tania Tudorache, Stanford University, USA
Solicited reviews: Ben De Meester, Ghent University, Belgium; Herminio Garcia-Gonzalez, Kazerne Dossin Research Centre, Belgium; Jose Emilio Labra Gayo, University of Oviedo, Spain

Abstract. Knowledge Graphs are currently created using an assortment of techniques and tools: ad hoc code in a programming language, database export scripts, OpenRefine transformations, mapping languages, etc. Focusing on the latter, the wide variety of use cases, data peculiarities, and potential uses has had a substantial impact in how mappings have been created, extended, and applied. As a result, a large number of languages and their associated tools have been created. In this paper, we present the Conceptual Mapping ontology, that is designed to represent the features and characteristics of existing declarative mapping languages to construct Knowledge Graphs. This ontology is built upon the requirements extracted from experts experience, a thorough analysis of the features and capabilities of current mapping languages presented as a comparative framework; and the languages' limitations discussed by the community and denoted as Mapping Challenges. The ontology is evaluated to ensure that it meets these requirements and has no inconsistencies, pitfalls or modelling errors, and is publicly available online along with its documentation and related resources.

Keywords: Mapping Languages, Ontology Description, Knowledge Graphs

1. Introduction

Data on the Web has steadily grown in the last decades. However, the heterogeneity of the data published on the Web has hindered its consumption and usage [1]. This scenario has fostered data transformation and publication of data as Knowledge Graphs in both academic and industrial environments [2]. These Knowledge Graphs normally expose Web data expressed in RDF and modeled according to an ontology.

A large number of techniques that query or translate data into RDF have been proposed, and follow two approaches, namely, (1) RDF materialization, that consists of translating data from one or more heterogeneous sources into RDF [3, 4]; or (2) Virtualization, (Ontology Based Data Access) [5, 6] that comprises translating a SPARQL query into one or more equivalent queries which are distributed and executed on the original data source(s), and where its results are transformed back to the SPARQL results format [7]. Both types of approaches rely on an essential element, a mapping document, which is the key-enabler for performing the required translation.

Mapping languages allow representing the relationships between the data model in heterogenous sources, and an RDF version that follows the schema of an

*Corresponding author. E-mail: ana.iglesiasm@upm.es.

ontology, i.e., they define the rules on how to translate from non-RDF data into RDF. The original data can be expressed in a variety of formats such as tabular, JSON, or XML. Due to the heterogeneous nature of data, the wide variety of techniques, and specific requirements that some scenarios may impose, an increasing number of mapping languages have been proposed [8–10]. The differences among them are usually based on three aspects: (a) the focus on one or more data formats, e.g., the W3C Recommendations R2RML focuses on SQL tabular data [11]; (b) a specific requirement they address, e.g., SPARQL-Generate [12] allows the definition of functions in a mapping for cleaning or linking the generated RDF data; or (c) if they are designed for a scenario that has special requirements, e.g., the WoT-mappings [13] were designed as an extension of the WoT standard [14] and used as part of the Thing Descriptions [15].

As a result, the diversity of mapping languages provides a rich variety of options for tools to translate data from heterogeneous formats into RDF, in many different scenarios [16–19]. However, these tools are mostly tied to one mapping language, and sometimes they do not even implement the entire language specification [4, 20]. Deciding which language and technique should be used in each scenario becomes a costly task, since the choice of one language may not cover all the needed requirements [21]. Some scenarios require a combination of mapping languages due to their different features, which requires the use of different techniques. In many cases, this diversity leads to ad hoc solutions that reduce reproducibility, maintainability, and reusability [22].

Mapping languages for KG construction maintain the same bottom-line idea and purpose: to describe and establish the relationships between data sources and the schema provided by an ontology. Therefore, it can be assumed that mapping languages share common inherent characteristics that can be modeled.

This paper presents the Conceptual Mapping ontology, which aims to gather the expressiveness of existing declarative mapping languages and represent their shared characteristics. The Conceptual Mapping ontology has been developed based on the requirements extracted from the Mapping Challenges proposed by the community[1] and the analysis of the features of state-of-the-art mapping languages. This analysis, presented as a comparative framework, studies how languages describe access to data sources, how they represent triples creation, and their distinctive features.

The Conceptual Mapping ontology has been developed following the LOT Methodology [23]. It reuses existing standards such as DCAT [24] and WoT Security[2]. The full mapping language specification is publicly available under the CC BY-SA 4.0 license. Several examples of usage, comparisons with other languages, extensions, and requirements are also available in the ontology portal[3].

The rest of this article is structured as follows. Section 2 provides an overview of relevant works centered on mapping languages. Section 3 describes the methodology used to develop the ontology. Section 4 presents the purpose and scope of the ontology, its requirements, and how they are extracted. Section 5 shows details about the ontology conceptualization and evaluation, and some examples. Section 6 illustrates how the ontology is published and maintained. Finally, Section 7 summarizes the work presented and draws some conclusions and future steps.

2. Related Work

In this section, the current scene of mapping languages is described first, regardless of the approach they follow, i.e., RDF materialization or virtualization. Then, previous works comparing mapping languages are surveyed.

2.1. Mapping languages

The different scenarios in which mapping languages are used and their specific requirements have led to the creation of several mapping languages and tailored to specific domain extensions. This section presents and describes existing mapping languages, listed in Table 1. Depending on their syntax, they can be classified into the following: RDF-based, SPARQL-based, and based on other schema. It is worth mentioning that some mapping languages have become W3C recommendations, namely R2RML [11] and CSVW [36]. The surveyed languages include the ones considered relevant because of their widespread use, unique features, and current maintenance. Deprecated or obsolete languages are not included.

[1] https://w3id.org/kg-construct/workshop/2021/challenges.html

[2] https://www.w3.org/2019/wot/security

[3] https://w3id.org/conceptual-mapping/portal

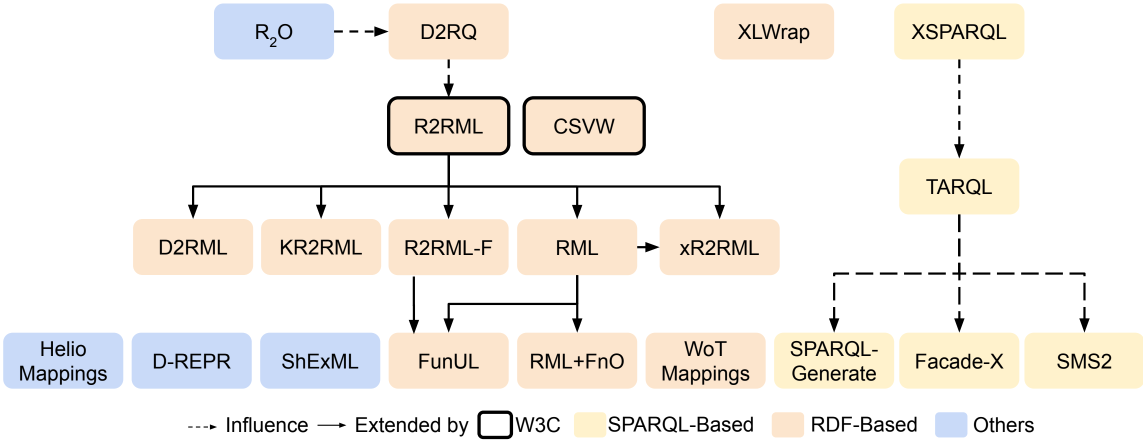


Fig. 1. Existing mapping languages and their relationships.

Table 1

Analyzed mapping languages and their corresponding references.

Classification	Language	Reference(s)
RDF-based	D2RQ	[25, 26]
	R_2O	[27]
	R2RML	[11]
	xR2RML	[9, 28]
	RML	[8, 29]
	KR2RML	[30]
	FunUL	[31]
	R2RML-f	[32]
	D2RML	[33]
	WoT mappings	[13]
	XLWrap	[34, 35]
	CSVW	[36]
SPARQL-based	SPARQL-Generate	[12, 37]
	XSPARQL	[38, 39]
	TARQL	[40]
	Facade-X	[41, 42]
	SMS2	[43]
Others	Helio mappings	[44]
	D-REPR	[45]
	ShExML	[10, 46, 47]
	XRM	[48]

RDF-based mapping languages. Similarly to Conceptual Mappings, these are mapping languages specified as ontologies. They are used as RDF documents that are processed by compliant tools for performing the translations. The evolution, extensions and influences on one another are depicted in Fig. 1. The most well-known language in this category is R2RML [11], which allows mapping of data stored in relational databases to RDF. This language is heavily influenced by previous languages (R_2O [27] and D2RQ [25]). Some serializations (e.g. SML [49], OBDA mappings from Ontop [50]) and several extensions of R2RML were developed in the following years after its release: R2RML-f [32] extends R2RML to include functions to be applied over the data; RML [8] and its user-friendly compact syntax YARRRML [51] provide the possibility of covering additional data formats (CSV, XML and JSON); this language also considers the use of functions for data transformation (e.g. lowercase, replace, trim) by using the Function Ontology (FnO)[4] [17]; FunUL [31] proposes an extension to also incorporate functions, but focusing on the CSV format; KR2RML [30] is also an extension for CSV, XML and JSON, with the addition of representing all sources with the Nested Relational Model as an intermediate model and the possibility of cleaning data with Python functions; xR2RML [9] extends R2RML and RML to include NoSQL databases and incorporates more features to handle tree-like data; D2RML [33], also based on R2RML and RML, is able to transform data from XML, JSON, CSVs and REST/SPARQL endpoints, and enables functions and conditions to create triples.

In this category, we can also find more languages not related to R2RML. XLWrap [34] is focused on transforming spreadsheets into different formats. CSVW [36] enables tabular data annotation on the Web with metadata, but also supports the generation of RDF. Finally, WoT Mappings [13] are oriented to be used in the context of the Web of Things.

[4] https://fno.io/rml/

SPARQL-based mapping languages. The specification of this type of languages is usually based on, or is an extension of, the SPARQL query language [52]. XSPARQL [38] merges SPARQL and XQuery to transform XML into RDF. TARQL [40] uses the SPARQL syntax to generate RDF from CSV files. SPARQL-Generate [12] is capable of generating RDF and document streams from a wide variety of data formats and access protocols. Most recently, Facade-X has been developed, not as a new language, but as a "*facade* to wrap the original resource and to make it queryable as if it was RDF" [41]. It does not extend the SPARQL language, instead it overrides the SERVICE operator. Lastly, authors would like to highlight a loosely SPARQL-based language, Stardog Mapping Syntax 2 (SMS2) [43], which represents virtual Stardog graphs and is able to support sources such as JSON, CSV, RDB, MongoDB and Elasticsearch.

Other mapping languages. This group gathers other mapping languages implemented without relying on ontologies or SPARQL extensions. ShExML [10, 46] uses Shape Expressions (ShEx) [53] to map data sources in RDBs, CSV, JSON, XML and RDF using SPARQL queries. The Helio mapping language [44] is based on JSON and provides the capability of using functions for data transformation and data linking [54]. D-REPR [45] focuses on describing heterogeneous data with JSONPath and allows the use of data transformation functions. XRM (Expressive RDF Mapper) [48] is a commercial language that provides a unique user-friendly syntax to create mappings in R2RML, CSVW and RML.

2.2. *Language comparison*

As the number of mapping languages increased and their adoption grew wider, comparisons between these languages inevitably occurred. This is the case of, for instance, SPARQL-Generate [12], which is compared to RML in terms of query/mapping complexity; and ShExML [10], which is compared to SPARQL-Generate and YARRRML from a usability perspective.

Some studies dig deeper, providing qualitative complex comparison frameworks. Hert et al. [55] provide a comparison framework for mapping languages focused on transforming relational databases to RDF. The framework is composed of 15 features, and the languages are evaluated based on the presence or absence of these features.The results lead authors to divide the mappings into four categories (direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose mapping), and ponder on the heavy reliance of most languages on SQL to implement the mapping, and the usefulness of read-write mappings (i.e., mappings able to write data in the database). De Meester et al. [21] show an initial analysis of 5 similar languages (RML+FnO, xR2RML, FunUL, SPARQL-Generate, YARRRML) discussing their characteristics, according to three categories: non-functional, functional and data source support. The study concludes by remarking on the need to build a more complete and precise comparative framework and asking for a more active participation from the community to build it. To the best of our knowledge, there is no comprehensive work in the literature comparing all existing languages.

3. Methodology

This section presents the methodology followed for developing the Conceptual Mapping ontology. The ontology was developed following the guidelines provided by the Linked Open Terms (LOT) methodology. LOT is a well-known and mature lightweight methodology for the development of ontologies and vocabularies that has been widely adopted in academic and industrial projects [23]. It is based on the previous NeOn methodology [56] and includes four major stages: Requirements Specification, Implementation, Publication, and Maintenance (Fig. 2). In this section, we describe these stages and how they have been applied and adapted to the development of the Conceptual Mapping ontology.

3.1. *Requirements specification*

This stage refers to the activities carried out for defining the requirements that the ontology must meet. At the beginning of the requirements identification stage, the goal and scope of the ontology are defined. Following, the domain is analyzed in more detail by looking at the documentation, data that has been published, standards, formats, etc. In addition, use cases and user stories are identified. Then, the requirements are specified in the form of competency questions and statements.

In this case, the specification of requirements includes purpose, scope, and requirements. The requirements are specified as facts rather than competency questions and validated with Themis [57], an ontology evaluation tool that allows validating requirements expressed as tests rather than SPARQL queries. The authors consider this approach to be adequate in this case since (1) there are no use cases as this ontology is a mechanism of representation of mapping language's features; and (2) there are no SPARQL queries because they result from Competency Questions which are in turn extracted from use cases and user stories. Further details are shown in Section 4.

3.2. Implementation

The goal of the Implementation stage is to build the ontology using a formal language, based on the ontological requirements identified in the previous stage. From the set of requirements a first version of the model is conceptualized. The model is subsequently refined by running the corresponding evaluations. Thus, the implementation process follows iterative sprints; once it passes all evaluations and meets the requirements, it is considered ready for publication.

The conceptualization is carried out representing the ontology in a graphical language using the Chowlk notation [58] (as shown in Fig. 4). The ontology is implemented in OWL 2 using Protégé. The evaluation checks different aspects of the ontology: (1) requirements are validated using Themis [57], (2) inconsistencies are found with the Pellet reasoner, (3) OOPS! [59] is used to identify modeling pitfalls, and (4) FOOPS! [60] is run to check the FAIRness of the ontology. Further details are described in Section 5.

3.3. Publication

The publication stage addresses the tasks related to making the ontology and its documentation available. The ontology documentation was generated with Widoco [61], a built-in documentation generator in OnToology [62], and it is published with a W3ID URL[5]. The ontology and related resources can be accessed in the ontology portal. Further details are presented in Section 6.

3.4. Maintenance

Finally, the last stage of the development process, maintenance, refers to ontology updates as new requirements are found and/or errors are fixed. The ontology presented in this work promotes the gathering of issues or new requirements through the use of issues in the ontology GitHub repository. Additionally, it provides control of changes, and the documentation enables access to previous versions. Further details are shown in Section 6.

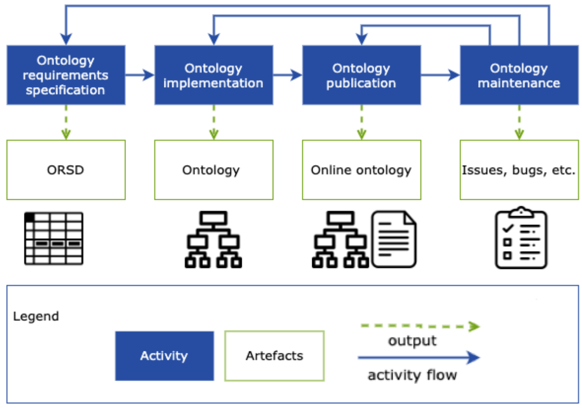


Fig. 2. Workflow proposed by the LOT Methodology [23].

4. Conceptual Mapping Requirements Specification

This section presents the purpose, scope, and requirements of the Conceptual Mapping Ontology. In addition, it also describes from where and how the requirements are extracted: analysing the mapping languages (presented as a comparative framework) and the Mapping Challenges proposed by the community.

4.1. Purpose and scope

The Conceptual Mapping ontology aims at gathering the expressiveness of declarative mapping languages that describe the transformation of heterogeneous data sources into RDF. This ontology-based language settles on the assumption that all mapping languages used for the same basic purpose of describing data sources in terms of an ontology to create RDF, must share some basic patterns and inherent characteristics. Inevitably, not all features are common. As described in previous sections, some languages were developed for specific purposes, others extend existing languages to cover additional use cases, and others are in turn based in languages that already provide them with certain capabilities. The Conceptual Mapping ontology is designed to represent and articulate these core features, which are extracted from two sources: (1) the analysis of current mapping languages, and (2) the lim-

[5] https://w3id.org/conceptual-mapping

itations of current languages identified by the community. These limitations, proposed by the W3C Knowledge Graph Construction Community Group[6], are referred to as Mapping Challenges[1] and have been partially implemented by some languages. Both sources are described throughout this section.

This ontology has also some limitations. As presented in Section 2, mapping languages can be classified into three categories according to the schema in which they are based: RDF-based, SPARQL-based and based on other schemes. Conceptual Mapping is included in the first category and, as such, has the same inherent capabilities and limitations as RDF-based languages regarding the representation of the language as an ontology. This implies that it is feasible to represent their expressiveness, whereas reusing classes and/or properties or creating equivalent constructs. Languages based on other approaches usually follow schemas that make them relatable to ontologies. This can be seen in the correspondence between YARRRML and RML: RML is written in Turtle syntax. YARRRML [51] is mainly used as a user-friendly syntax to facilitate the writing of RML rules. It is based on YAML, and can easily be translated into RML[7].

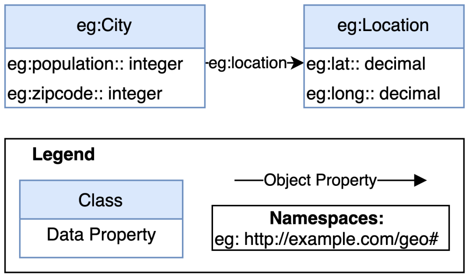
Lastly, SPARQL-based languages pose a challenge. SPARQL is a rich and powerful query language [63] to which these mapping languages add more capabilities (e.g., SPARQL-Generate, Facade-X). It has an innate flexibility and capabilities sometimes not comparable to the other languages. For this reason, representing every single capability and feature of SPARQL-based languages is out of the scope of this article. Given the differences of representation paradigm between RDF and SPARQL for creating mappings, it cannot be ensured that the Conceptual Mapping covers all possibilities that a SPARQL-based language can.

4.2. Comparison Framework

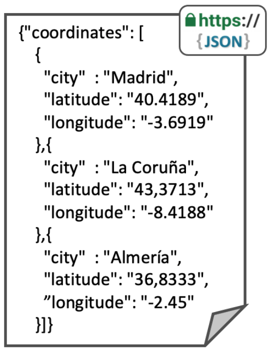
This subsection presents a comparison framework that collects and analyzes the main features included in mapping language descriptions. It aims to fill the aforementioned gap on language comparison. The diversity of the languages that have been analyzed is crucial for extracting relevant features and requirements. For this reason, the framework analyzes languages from the three categories identified in Section 2.

[6]https://www.w3.org/community/kg-construct/

[7]https://rml.io/yarrrml/matey/



(a) Example reference ontology that represents the classes `City` and `Location`, linked by the property `eg:location`.



(b) Example input JSON file "coordinates.json".

city	population	year_modified	zipcodes
A Coruña	244850	2018	15001, 15002, 15003, 15004
Almeria	201322	2021	04001, 04002
Madrid	3334730	2021	28001, 28002, 28003, 28004, 28005, 28006

(c) Example input MySQL table "cities".

Fig. 3. Input source data and reference ontology that represents information on cities and their location.

The selected languages fulfill the following requirements: (1) widely used, relevant and/or include novel or unique features; (2) currently maintained, and not deprecated; (3) not a serialization or a user-friendly representation of another language. For instance, D2RQ [25] and R_2O [27] were superseded by R2RML, which is included in the comparison. XRM [48] is not included either, due to the fact that it provides a syntax for CSVW, RML and R2RML, which are also included.

The following RDF-based languages are included: R2RML [11], RML [8], KR2RML [30], xR2RML [9], R2RML-F [32], FunUL [31], XLWrap [34], WoT mappings [13], CSVW [36], and D2RML [33]. The SPARQL-based languages that were analyzed are: XSPARQL [38], TARQL [40], SPARQL-Generate [12], Facade-X [41] and SMS2 [43]. Finally, we selected the following languages based on other formats: ShExML [10], Helio Mappings [44] and D-REPR [45].

These languages have been analyzed based on their official specification, documentation, or reference paper (listed in Table 1). Specific implementations and extensions that are not included in the official documentation are not considered in this framework. The cells (i.e. language feature) marked "*" in the framework tables indicate that there are non-official implementations or extensions that include the feature.

The framework has been built as a result of analyzing the common features of the aforementioned mapping languages, and also the specific features that make them unique and suitable for some scenarios. It includes information on data sources, general features for the construction of RDF graphs, and features related to the creation of subjects, predicates, and objects. In the following subsections, the features of each part of the framework are explained in detail. The language comparison for data sources is provided in Table 2, for triples creation in Table 3, and for general features in Table 4. All these tables are presented in Appendix B.

Throughout the section, there are examples showing how different languages use the analyzed features. The example is built upon two input sources: an online JSON file, "coordinates.json", with geographical coordinates (Fig. 3b); and a table from a MySQL database, "cities" (Fig. 3c). The reference ontology is depicted in Fig. 3a. It represents information about cities and their locations. The expected RDF output of the data transformation is shown in Listing 1. Each mapping represents only the relevant rules that the subsection describes. The entire mapping can be found in the examples section of the ontology documentation[5].

```
<http://ex.com/loc/40.4189—3.6919> a eg:Location ;
  eg:lat "40.4189"^^xsd:decimal ;
  eg:long "-3.6919"^^xsd:decimal .

<http://ex.com/loc/43.3713—8.4188> a eg:Location ;
  eg:lat "43.3713"^^xsd:decimal ;
  eg:long "-8.4188"^^xsd:decimal .

<http://ex.com/loc/36.8333—2.45> a eg:Location ;
  eg:lat "36.8333"^^xsd:decimal ;
  eg:long "-2.45"^^xsd:decimal .

<http://ex.com/city/ACoruña> a eg:City ;
  eg:zipcode 15001, 15002, 15003, 15004 ;
  eg:location <http://ex.com/loc/43.3713—8.4188> .

<http://ex.com/city/Almería> a eg:City ;
  eg:zipcode 04001, 04002 ;
  eg:population 201322 ;
  eg:location <http://ex.com/loc/36.8333—2.45> .

<http://ex.com/city/Madrid> a eg:City ;
  eg:zipcode 28001, 28002, 28003, 28004, 28005, 28006;
  eg:population 3334730 ;
  eg:location <http://ex.com/loc/40.4189—3.6919> .
```

Listing 1: Expected RDF output for the data sources and the ontology in Fig. 3.

4.2.1. Data Sources Description

Table 2 shows the ability of each mapping language to describe a data source in terms of retrieval, features, security, data format and protocol.

Data Retrieval. Data from data sources may be retrieved in a continuous manner (e.g., *Streams*), periodically (e.g., *Asynchronous sources*), or just once, when the mapping is executed (e.g., *Synchronous sources*). As shown in Table 2, all mapping languages are able to represent synchronous data sources. Additionally, SPARQL-Generate and Helio are able to represent periodical data sources, and SPARQL-Generate also represents continuous data sources (e.g. `it:WebSocket()` in SPARQL-Generate). Other languages do not explicitly express that feature in the language, but a compliant engine may implement it.

Representing Data Sources. Extracting and retrieving heterogeneous data involves several elements that mapping languages need to consider: *Security terms* to describe access (e.g., relational databases (RDB), API Key, OAuth2, etc); *Retrieval protocol* such as local files, HTTP(S), JDBC, etc; *Features that describe the data* to define particular characteristics of the source data (e.g. queries, regex, iterator, delimiter, etc); *Data formats* such as CSV, RDB, and JSON; *Encoding* and content negotiation (i.e. *MIME Type*).

Half of the languages do not allow the definition of security terms. Some languages are specific for RDB terms (R2RML and extensions, with `rr:logicalTable`), and only two, Helio and WoT, can define security terms. These two languages are also the only ones that allow the specification of MIME Types, and can also specify the encoding along with

TARQL and CSVW (e.g. `csvw:encoding` attribute of `csvw:Dialect` in CSVW).

Regarding protocols, all languages consider local files, except WoT mappings, which are specific for HTTP(s). It is highly usual to consider HTTP(s) and database access (especially with the ODBC and JDBC protocols). Only XSPARQL, TARQL, D-REPR, and XLWrap describe exclusively local files.

The features provided by each language are closely related to the data formats that are covered. Queries are usual for relational databases and NoSQL document stores and iterators for tree-like formats. Some languages also enable the description of delimiters and separators for tabular formats (e.g., CSVW defines the class `Dialect` to describe these features; this class is reused by RML), and finally, less common Regular Expressions can be defined to match specific parts of the data in languages such as CSVW, SPARQL-Generate, Helio, D-REPR, and D2RML (e.g., `RegexHandler` in Helio, `format` in CSVW).

The most used format is tabular (RDB and CSV). Some languages can also process RDF graphs such as SMS2, ShExML, RML, SPARQL-Generate, Helio, and D2RML (e.g. `QUERY` in ShExML, SPARQL service description[8] in RML), and the last three languages can also process plain text.

Data Sources Example. This example shows how ShExML and R2RML describe heterogeneous data sources. The sources are a table called "cities" (Fig. 3c) that belongs to a relational database that stores information about cities: name, population, zipcode and year in which the data was updated; and a JSON file "coordinates.json" (Fig. 3b) available online that contains the latitude and longitude of the central point of each city. R2RML is only able to describe the database table (Listing 3); instead ShExML is able to describe both the RDB and the online JSON file (Listing 3).

```
<#CitiesSource> a rr:LogicalTable;
  rr:tableName "cities" .
```

Listing 2: R2RML mapping file describing Fig. 3b and Fig. 3c.

```
SOURCE cities_rdb <jdbc:mysql://localhost:3306/citydb>
SOURCE coord_json <https://ex.com/geodata/coordinates.json>
ITERATOR it_cities <sql: SELECT * FROM cities;> {
   FIELD c_city <city>
   FIELD population <population>
   FIELD year <year_modified>
   FIELD zipcode <zipcodes>
}
ITERATOR it_coord <jsonpath: $.coordinates[*]> {
   FIELD lat <latitude>
   FIELD long <longitude>
   FIELD loc_city <city>
}
```

Listing 3: ShExML mapping file describing Fig. 3b and Fig. 3c.

4.2.2. Triples Generation

Table 3 represents how different languages describe the generation of triples. We assess whether they generate the *Subject*, *Predicate*, and *Object*: in (1) a *Constant* manner, i.e. non-dependant on the data field to be created; or in (2) a *Dynamic* manner, i.e. changing its value with each data field iteration. For *Objects*, the possibility of adding *Datatype and Language* tags is also considered; this feature assesses whether they can be added, and if they are added in a dynamic (changes with the data) or static (constant) manner. This table also analyzes the use and cardinality of transformation functions and the possibility of iterating over different nested level arrays (i.e., in tree-like formats).

The categories *Constant* and *RDF Resource* (the latter within *Dynamic*) show which kind of resources can be generated by the language (i.e., IRI, Blank Node, Literal, List and/or Container). The *Dynamic* category also considers: the *Data References* (i.e. fields from the data source) that can appear with single of mixed formats; from how many *Data Sources* (e.g. "1:1" when only data from one file can be used) the term is generated; if *Hierarchy Iteration* over different nested levels in tree-like formats is allowed; and if *Functions* can be used to perform transformations on the data to create the term (e.g. lowercase, toDate, etc.).

Subject Generation. Subjects can be IRIs or Blank Nodes (BN). This is well reflected in the languages, since, with a few exceptions that do not consider Blank Nodes, all languages are able to generate these two types of RDF resources, both constant and dynamically. The WoT mappings can only generate constant subjects, so the dynamic dimensions do not apply to this language. The rest of the languages can generate a subject with one or more data references (e.g., in RML `rr:template "http://ex.org/{id}-`

[8] http://www.w3.org/ns/sparql-service-description#

`{name}`"), ShExML, xR2RML, SPARQL-Generate, Facade-X, and Helio with different formats. For example, in xR2RML a CSV field that contains an array can be expressed as: `xrr:reference "Column(Movies)/JSONPath($.*)`. Part of the languages even allow generating subjects with more than one data source, this is the case of ShExML, XSPARQL, KR2RML, SPARQL-Generate, Facade-X, Helio and xR2RML. About a third of the languages allow hierarchy iterations (ShExML, XSPARQL, KR2RML, SPARQL-Generate, D-REPR, Facade-X, SMS2, and D2RML), and more than a half use functions with N:1 cardinality. Additionally, some of them even allow functions that can output more than one parameter (i.e., 1:N or N:M), but it is less usual.

Predicate Generation. All languages can generate constant predicates as IRIs. Only four languages do not allow dynamic predicates (WoT mappings, SMS2, ShExML, and XLWrap). For those that do, they also allow more than one data reference. The languages that allow subject generation using multiple formats, data sources, functions, and hierarchy iterations, provide the same features for predicate generation.

Object Generation. Generally, languages can generate a wider range of resources for objects, since they can be IRIs, blank nodes, literals, lists, or containers. All of them can generate constant and dynamic literals and IRIs. Those languages that allow blank nodes in the subject also allow them in the object. Additionally, ShExML, KR2RML, SPARQL-Generate, Facade-X, xR2RML, and WoT mappings consider lists, and the last two languages also consider containers (e.g. `rr:termType xrr:RdfBag` in xR2RML). Data references, sources, hierarchy iterations, and functions remain the same as in subject generation, with the addition of WoT mappings that allow dynamic objects. Lastly, datatype and language tags are not allowed in KR2RML and XLWrap; they are defined as constants in the rest of the languages, and dynamically in ShExML, XSPARQL, TARQL, RML, and Helio (e.g., `rml:languageMap` for dynamic language tags in RML).

Triples Generation Example. Assuming the description of the data sources shown in Fig. 3b and Fig. 3c, this example illustrates how xR2RML and RML+FnO describe the rules to generate triples according to the ontology depicted in Fig. 3a. Instances of the classes `eg:City` and `eg:Location` have to be created, along with values for the attributes `eg:lat`, `eg:long` and `eg:zipcode`. A function is required to remove the spaces in the field "city" from the database table (Fig. 3c) in order to create the URI of the instances correctly. In addition, the field "zipcodes" has to be separated to retrieve each of its values (see expected output in Listing 1). xR2RML is capable of correctly generating zip codes (Listing 5), but it lacks the ability to correctly generate URI without spaces. RML+FnO is capable of doing the opposite (Listing 4).

```
mappings:
  Locations:
    sources: coord-source
    s: http://ex.com/loc/$(latitude)-$(longitude)
    po:
      - [rdf:type, eg:Location]
      - [eg:lat, $(latitude), xsd:decimal]
      - [eg:long, $(longitude), xsd:decimal]

  Cities:
    sources: cities-source
    s:
      - function: fun:concat
        parameters:
          - [fun:param1, "http://ex.com/city/"]
          - parameter: fun:param2
            value:
              function: fun:replace
              parameters:
                - [fun:param1, $(city)]
                - [fun:param2, " "]
                - [fun:param3, ""]
    po:
      - [rdf:type, eg:City]
      - [eg:zipcode, $(zipcodes), xsd:integer]
```

Listing 4: RML+FnO mapping rules (written in YARRRML) to describe the ontology depicted in Fig. 3a.

```
<#Locations> a rr:TriplesMap ;
 xrr:logicalSource <#LocationSource> ;
 rr:subjectMap [
  rr:template "http://ex.com/loc/{$.latitude}-{$.
      longitude}" ;
  rr:class eg:Location;];
 rr:predicateObjectMap [
  rr:predicate eg:lat ;
  rr:objectMap [ xrr:reference "$.latitude";
    rr:datatype xsd:decimal];];
 rr:predicateObjectMap [
   rr:predicate eg:long ;
   rr:objectMap [ xrr:reference "$.longitude";
     rr:datatype xsd:decimal];].
```

```
<#Cities> a rr:TriplesMap ;
 xrr:logicalSource <#CitiesSource> ;
 rr:subjectMap [
   rr:template "http://ex.com/city/{city}" ;
   rr:class eg:City ; ];
 rr:predicateObjectMap [
   rr:predicate eg:zipcode ;
   rr:objectMap [
    xrr:reference "Column(zipcodes)/JSONPath($.*)";
    rr:datatype xsd:integer] ;].
```

Listing 5: xR2RML mapping rules to describe the ontology depicted in Fig. 3a.

4.2.3. General Features for Graph Construction

Table 4 shows the features of mapping languages regarding the construction of RDF graphs such as *linking rules*, *metadata* or *conditions*, assignment to *named graphs*, and declaration of *transformation functions* within the mapping.

Statements. General features that apply to statements are described in this section: the capability of a language to assign statements to *named graphs*, to *retrieve data from only one source* or *more than one source*, and to apply *conditions* that have to be met in order to create the statement (e.g. if the value of a field called "required" is `TRUE`, the triple is generated).

Most RDF-based languages allow static assignment to named graphs. R2RML, RML, R2RML-F, FunUL, and D2RML enable also dynamic definitions (e.g., `rr:graphMap` in R2RML and in its extensions mentioned above). Theoretically, the rest of R2RML extensions should also implement this feature; however, to the best of our knowledge, it is not mentioned in their respective specifications.

Allowing conditional statements is not usual; it is only considered in the SPARQL-based languages (with the exception of SMS2), XLWrap and D2RML (e.g. `xl:breakCondition` in XLWrap). Regarding data sources, all languages allow data retrieval from at least one source; ShExML, XSPARQL, CSVW, SPARQL-Generate, Facade-X, Helio, D-REPR and D2RML enable more sources. That is, using data in the same statement from, e.g., one CSV file and one JSON file.

Linking Rules. Linking rules refer to linking resources that are being created in the mapping. For instance, having as object of a statement a resource that is the subject of another statement. These links are implemented in most languages by joining one or more data fields. Six languages do not allow these links: TARQL, CSVW, KR2RML, WoT, SMS2, and XLWrap. The rest is able to perform linking with at least one data reference and one or no condition. Fewer enable more data references and more conditions (e.g. in R2RML and most extensions allow the application of a `rr:joinCondition` over several fields).

Linking rules using join conditions imply evaluating if the fields selected are equal. Since the join condition is the most common, applying the equal logical operator is the preferred choice. Only a few languages consider other similarity functions to perform link discovery, such as the Levenshtein distance and Jaro-Winkler, e.g., Helio.

Transformation functions. Applying functions in mappings allows practitioners transforming data before it is translated. For instance, to generate a label with an initial capital letter (`ex:ID001 rdfs:label "Emily"`) that was originally in lower case ("emily"), a function may be applied (e.g. GREL function `toTitleCase()`). Only four of the analyzed languages do not allow the use of these functions: CSVW, R2RML, xR2RML, and WoT mappings. Of those that do, some use functions that belong to a specification (e.g. RML+FnO uses GREL functions[9]). All of them consider functions with cardinalities 1:1 and N:1; and half of them also include 1:N and N:M (i.e., output more than one value), for instance, a regular expression that matches and returns more than one value. Nesting functions (i.e. calling a function inside another function) is not unusual; this is the case of SPARQL-based languages, the R2RML extensions that implement functions (except K2RML), Helio, D-REPR, and XLWrap. Finally, some languages even enable extending functions depending on specific user needs, such as XSPARQL, RML+FnO, SPARQL-Generate, Facade-X, R2RML-F, FunUL, XLWrap and D2RML.

Graph Construction Example. Assuming the description of data sources shown in Fig. 3b and Fig. 3c and the regular triples, this example shows how Helio and SPARQL-Generate describe conditional statements and linking rules. To generate the `eg:population` attribute (Fig. 3a), the record must have been updated after 2020. In addition, instances of the classes `eg:City` and `eg:Location` can be joined using the city name, present in both data sources. However, the names do not exactly match ("Almería" and

[9] https://docs.openrefine.org/manual/grelfunctions

"Almeria"; "A Coruña" and "La Coruña"), which is why a distance metric is required to match the cities with a threshold of 0.75. The Helio mapping is not capable of describing the condition of the population, but instead it is able to use the Levenshtein distance function and link the sources (Listing 7). SPARQL-Generate can describe the condition statement thanks to the SPARQL construct `FILTER`, but does not implement the distance metric function (Listing 6). However, both Helio and SPARQL-Generate allow the removal of spaces in the subject URIs.

```
GENERATE {
  <city/{REPLACE( ?city, " ", "")}> a eg:City .
  <loc/{?lat}-{?long}> a eg:Location .

  GENERATE {
    <city/{REPLACE( ?city, " ", "")}> eg:population ?population.
  } WHERE {
    FILTER("{?year_modified}"^^xsd:integer > 2020)}.

  GENERATE {
    <city/{REPLACE( ?city, " ", "")}> eg:location <loc/{?lat}-{?long}>.
  } WHERE {
    FILTER(?loc_city = ?city)}.
}
```

Listing 6: SPARQL-Generate query with conditional rules to describe the ontology depicted in Fig. 3a.

```
{"resource_rules" : [
  {
    "id" : "Locations",
    "datasource_ids" : ["locations_source"],
    "subject" : "http://ex.com/loc/{$.latitude}-{$.longitude}",
  },{
    "id" : "Cities",
    "datasource_ids" : ["cities_source"],
    "subject" : "http://ex.com/city/[replace({$.city}, ' ', '')]",
    "properties" : [{
      "predicate" : "http://example.com/geo#population",
      "object" : "{population}",
      "is_literal" : "True",
    }]
  }],
 "link_rules" : [
  {
    "condition" : "levenshtein(S({city}), T({$.city}))>0.75",
    "source" : "Cities",
    "target" : "Locations",
    "predicate" : "http://example.com/geo#location"
  }]}
```

Listing 7: Helio mapping with linking rules to describe the ontology depicted in Fig. 3a.

4.3. Mapping Challenges

Following its inception, the W3C Knowledge Graph Construction Community Group[6] defined a series of challenges for mapping languages based on the experience of members in using declarative mappings[1]. These challenges are a summary of the limitations of current languages. They have been partially addressed independently in some of the analyzed languages, such as RML [64] and ShExML [46]. These challenges are summarized as follows:

- **[C1] Language Tags and Datatype.** It refers to dynamically building language tags ([C1a]) and datatypes ([C1b]), that is, from data rather than as constant values.
- **[C2] Iterators.** This challenge addresses the need to access data values 'outside' the iteration pattern ([C2a]), especially in some tree-like data sources such as JSON; and iterating over multi-value references ([C2b]).
- **[C3] Multi-value References.** It discusses how languages handle data fields that contain multiple values ([C3a]), their datatypes and associated language tags ([C3b]).
- **[C4] RDF Collections and Containers.** This challenge addresses the need to handle RDF collections and containers.
- **[C5] Joins.** It refers to joining resources with zero join conditions ([C5a]) and joining literals instead of IRIs ([C5b]).

4.4. Conceptual Mapping Requirements

In order to extract the requirements that serve as the basis for the development of the Conceptual Mapping ontology, we take as input the analysis from the comparison framework and the Mapping Challenges described in previous sections. From a combination of their features, we extract 30 requirements. These requirements are expressed as facts, and are available in the ontology repository and portal[10]. Each requirement has a unique identifier, its provenance (comparison framework or mapping challenge id) and the corresponding constructs in the ontology. The constructs are written in Turtle, and lack cardinality restrictions for the sake of understandability. These requirements are tested with Themis, and its corresponding tests include these restrictions. More details on the evaluation of the requirements are provided in Section 5.3.

The requirements gathered range from general-purpose to fine-grained details. The general-purpose requirements refer to the basic fundamental capabilities of mappings, e.g., to create the rules to generate RDF triples (cm-r8) from reference data sources (cm-r7). The requirements with the next level of detail involve some specific restrictions and functionalities, e.g. to indicate the specific type (whether they are IRIs, Blank nodes, etc.) of subjects (cm-r16), predicates (cm-r17), objects (cm-r18), named graphs (cm-r19), datatypes (cm-r20) and language tags (cm-r21); the possibility of using linking conditions (cm-r23) and functions (cm-r15). Finally, some requirements refer to specific details or features regarding the description of data sources (e.g. cm-r4, cm-r6) and transformation rules (e.g. cm-r14, cm-r22, cm-r25).

Not all the observed features in the comparison framework have been added to the set of requirements. Some features are really specific, and supported by a minority of languages, sometimes only one language. As a result, we selected the (really) detailed features in these requirements to build the core specification of the Conceptual Mapping when they tackled the basic functionalities of the language. The rest of the details are left to be included as extensions. This differentiation and the modeling criteria is explained further in Section 5.

5. Conceptual Mapping Implementation

This section describes in detail the activities and tasks carried out to implement the ontology, that consists in the conceptualization of the model, the encoding in a formal language, and the evaluation to fix errors, inconsistencies, and ensure that it meets the requirements. Additionally, an example of the ontology's use is presented at the end of the section.

[10]https://oeg-upm.github.io/Conceptual-Mapping/requirements/requirements-core.html

5.1. Ontology Conceptualization

The ontology's conceptualization is built upon the requirements extracted from experts experience, a thorough analysis of the features and capabilities of current mapping languages presented as a comparative framework; and the languages' limitations discussed by the community and denoted as Mapping Challenges. The resulting ontology model is depicted in Fig. 4. This model represents the core specification of the Conceptual Mapping ontology that contains the essential features to cover the requirements. Some detailed features are also included when considered important to the language expressiveness, or needed for the language main functionality. Other detailed features are considered as extensions, as explained further in this section. For description purposes, we divide the ontology into two parts, Statements and Data Sources, that compose the core model. These two parts, when not used in combination, cannot describe a complete mapping. For that reason they are not separated into single modules.

Data sources. A data source (`DataSource`) describes the source data that will be translated. For this section, the Data Catalog (DCAT) vocabulary [24] has been reused. `DataSource` is a subclass of `dcat:Distribution`, which is a specific representation of a dataset (`dcat:Dataset`), defined as "data encoded in a certain structure such as lists, tables and databases". A source can be a streaming source (`StreamSource`) that continuously generates data, a synchronous source (`SynchronousSource`) or an asynchronous source (`AsynchronousSource`). Asynchronous sources, in turn, can be event sources (`EventSource`) or periodic sources (`PeriodicSource`). The details of the data source access are represented with the data access service class (`DataAccessService`), which in turn is a subclass of `dcat:DataService`. This class represents a collection of operations that provides access to one or more datasets or data processing functions, i.e., a description of how the data is accessed and retrieved. The data access service optionally has a security scheme (e.g., OAuth2, API Key, etc.) and an access protocol (e.g., HTTP(s), FTP, etc.).

Data properties in the `dcat:Dataset`, `dcat:Distribution` and `dcat:DataService` classes may be reused according to the features that may be represented in each mapping language, e.g. `dcat:endpointDescription`, `dcat:endpointURL`

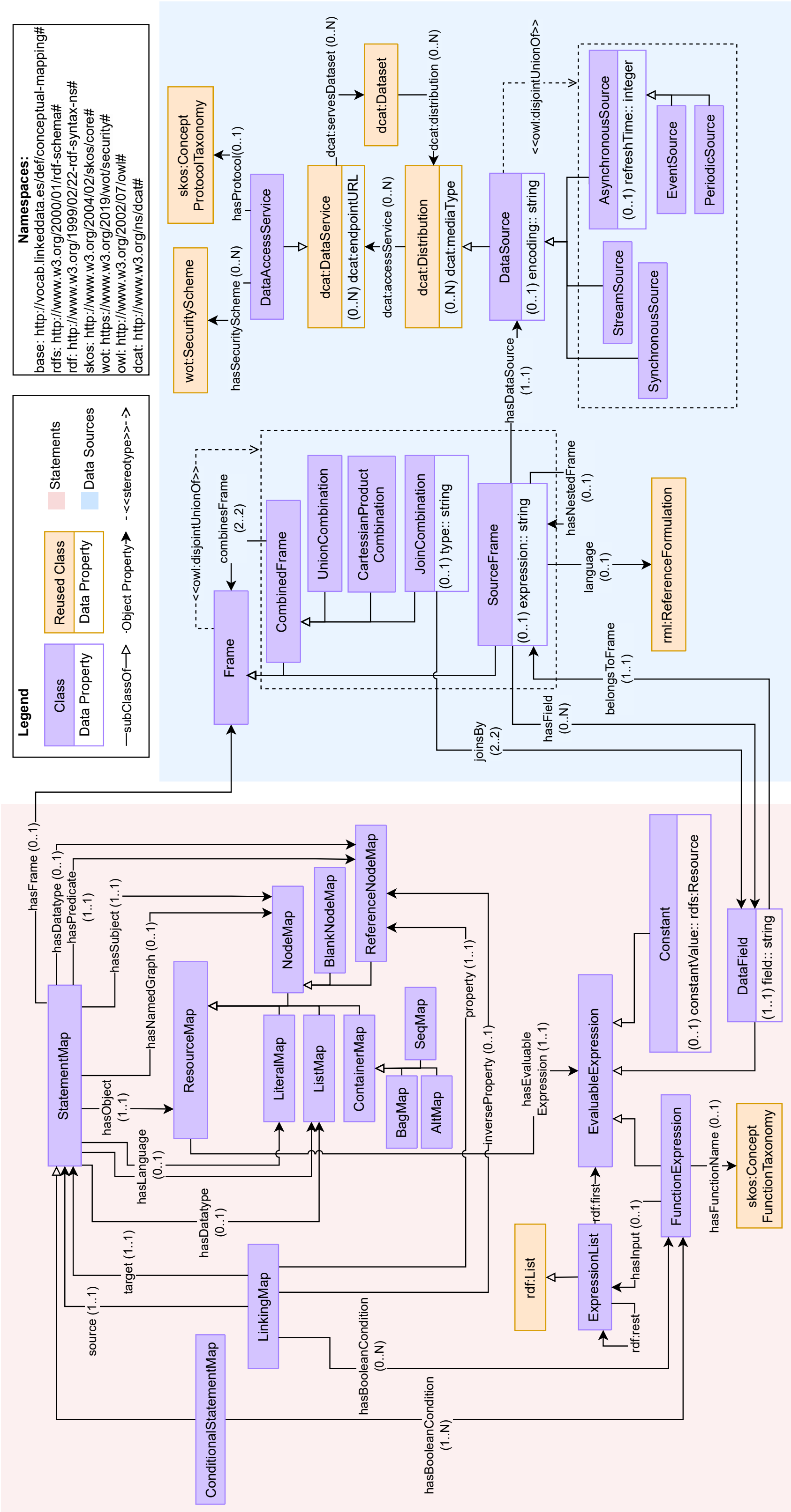


Fig. 4.: Visual representation of the Conceptual Mapping ontology created using the Chowlk diagram notation [58].

and `dcat:accessURL`. A data access service is related to a security scheme. The class `wot:SecurityScheme` (from the Web of Things (WoT) Security ontology[2]) has been reused. This class has different types of security schemes as subclasses and includes properties to specify the information on the scheme (e.g. the encryption algorithm, the format of the authentication information, the location of the authentication information). The security protocol `hasProtocol` has as set of predefined values that have been organized as a SKOS concept scheme. It contains almost 200 security protocols, e.g., HTTP(s), JDBC, FTP, GEO, among others. This SKOS list can be extended according to the users' needs by adding new concepts.

In order to represent the fragments of data that are referenced in a statement map, the class `Frame` has been defined. They are connected with the property `hasFrame`. A frame can be a `SourceFrame` (base case) or a `CombinedFrame`, the latter representing two source frames or combined frames that are combined by means of a join (`JoinCombination`), a union (`UnionCombination`) or a cartesian product (`CartesianProductCombination`).

A source frame corresponds to a data source (with `hasDataSource`) and defines which data is retrieved from the source and how it is fragmented (with `expression`). Among others, JSONPaths, XPaths, queries, or regular expressions can be expressed with this feature. The language of the expression is defined with `language`, which domain is the reused class from RML `rml:ReferenceFormulation`. A source frame may be related to another source frame with `hasNestedFrame`, e.g. a frame is accessed firstly with a SPARQL query, and their results as a CSV file with this property. A source fragment may refer to many data fields (with `hasField`, which is the inverse property of `belongsToFrame`).

Statements. The central class of this section is the `StatementMap`, which represents a rule that defines for a triple its subject (`hasSubject`), predicate (`hasPredicate`), and object (`hasObject`). Optionally, it can also specify the object datatype (`hasDatatype`), language (`hasLanguage`) and assigned named graph (`hasNamedGraph`). Therefore, statement maps are similar to RDF statements as both of them are comprised by a subject, predicate and object. In statement maps, objects are resources (`ResourceMap`), and subjects and predicates are more specific, certain subclasses of the resource map: predicates are reference node maps (`ReferenceNodeMap`) that represent resources with an IRI, i.e., ontology properties. Subjects are node maps (`NodeMap`) that may be blank nodes (`Blank Node`) or also reference node maps. An object may be a literal (`LiteralMap`), a blank node, a container (`ContainerMap`) or a collection that defines a list (`ListMap`). The language is expressed as a literal, and the datatype is also a resource with an IRI, i.e. a reference node map.

Resource maps are expressed with an evaluable expression (`EvaluableExpression`) that may be a constant value (`Constant`), a function expression (`FunctionExpression`), or a data field (`DataField`) that belongs to some data source fragment (`belongsToFrame`). For function expressions, the function name (`hasFuntionName`) is taken from a set of predefined names organized in a SKOS concept scheme. This SKOS list can be extended according to the users' needs by adding new concepts for functions that have not been defined. Recursion in this function expression is represented through its input (`hasInput`) as an expression list (`ExpressionList`). Expression lists have been represented as a subclass of RDF lists (`rdf:List`), and the properties (`rdf:first`) and (`rdf:rest`) have been reused. Expression lists may have nested expression lists inside.

A special case of a statement map is a conditional statement map (`ConditionalStatementMap`), a statement map that must satisfy a condition for the triples to be generated. The condition (`hasBoolean Condition`) is a function expression (e.g. if a value from a field called "present" is set to "False", the statement is not generated). Another relevant class is the linking map (`LinkingMap`), that enables linking subjects from a source (`source`) and a target (`target`) statement maps, i.e., two resources are linked and triples are generated if a linking condition is satisfied. Similarly to the conditional statement map, this condition is represented as a function expression.

5.2. *Ontology Design Patterns*

The following ontology design patterns have been applied in the conceptualization as they are common solutions to the problem of representing taxonomies and linked lists:

- The SKOS vocabulary has been reused to represent some coding schemes such as the proto-

col taxonomy and the function taxonomy. The design pattern consists on having an instance of `skos:ConceptScheme` for each taxonomy, then each concept or term in the taxonomy, `skos:Concept`, is related to the corresponding concept scheme through the property `skos:inScheme`. The class that uses the taxonomy is then related to `skos:Concept` through an object property, e.g., class `DataAccessService` and object property `hasProtocol`.
- The class `ExpressionList` uses the design pattern for lists developed in RDF where the properties `rdf:first` and `rdf:rest` are used to represent a linked list. The base case (first) is an evaluable expression whereas the rest of the list is (recursively) an `ExpressionList`.

5.3. Ontology evaluation

The ontology, once implemented, has been evaluated in different ways to ensure that it is correctly implemented, it has no errors or pitfalls, and meets the requirements.

Reasoner. We used the reasoner Pellet in Protégé to look for inconsistencies in the model, and the results showed no errors.

OOPS!. This tool was used to identify modeling pitfalls in the ontology. We executed the tool several times to fix the pitfalls, until there were no important ones. Currently, the results of OOPS! show pitfalls from the reused ontologies, but none important for the newly created terms and axioms. One minor pitfall is returned, P13, regarding the lack of inverse relationships, which we consider that are not needed in the ontology. The rest of the pitfalls are as follows: P08 (missing annotations) from DCTERMS; P11 (missing domain or range in properties) for DCTERMS, DCAT and SKOS; and P20 (misusing ontology annotations) for DCAT.

Themis. Themis is able to evaluate whether the requirements are implemented in the ontology. To that end, the requirements must be provided in a specific syntax or described with the Verification Test Case (VTC) ontology[11]. The requirements of the Conceptual Mapping were translated to create the corresponding tests, and were tested in the tool with success. The requirements and associated test along with the complete set of tests annotated with the VTC ontology are available in the GitHub repository[12].

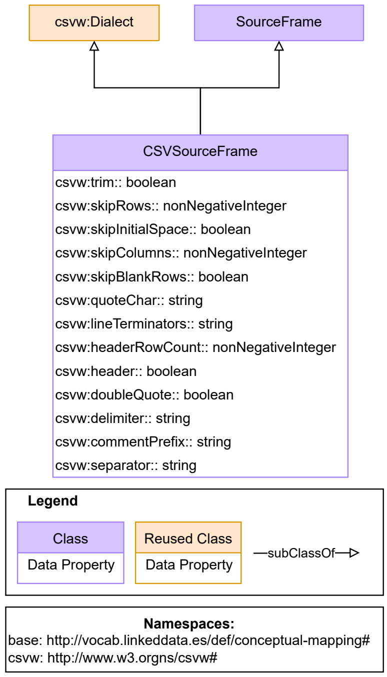


Fig. 5. CSV extension conceptualization.

FOOPS!. Additionally, we tried running FOOPS! to check the FAIRness of the ontology, resulting in 73%, which is acceptable. To improve the score, the ontology should be added to a registry and have more metadata describing it, and use a persistent base IRI.

With these evaluations, we can conclude that the ontology is correctly encoded and implemented, and that it meets the requirements specified in Section 4.

5.4. Extensibility

The Conceptual Mapping ontology has been designed as a core ontology. However, as time passes, new requirements may emerge. In order to include these new requirements, new modules of the Conceptual Mapping ontology shall be developed. It is worth mentioning that this is a common practice for ontologies, which is highly suitable for adapting an existing ontology to new scenarios, by ontology modules

[11] https://albaizq.github.io/test-verification-ontology/OnToology/ontology/verification-test-description.ttl/documentation/index-en.html

[12] https://github.com/oeg-upm/Conceptual-Mapping/tree/main/requirements

specialized for a specific set of requirements. A clear example of this is the SAREF ontology[13], that has a core module[14] and then specific extensions[15] for certain domains, such as energy (SAREF4ENER), buildings (SAREF4BLDG), etc.

In the case of the Conceptual Mappings a sample extension[16] is provided to showcase this feature. The extension focuses on describing CSV, a detailed feature present in some languages but not included in the core specification presented in previous sections. To this end, the CSVW proposal has been blended as an ontology module linked to the core Conceptual Mapping ontology. This module is depicted in Fig. 5.

5.5. *Ontology usage example*

This section builds a mapping in three steps (data sources in Listing 8, triples in Listing 9 and special statements in Listing 10) to represent how the proposed language can describe data with different features. The mapping uses the data sources "coordinates.json" (Fig. 3b) and "cities"(Fig. 3c) as input and the ontology depicted in Fig. 3a as reference, to create the output RDF shown in Listing 1. Additionally, Appendix A contains a second example to illustrate different features than the ones represented in the example of this section, to provide more insights about the expressiveness of this language.

Data sources. Listing 8 shows the description of the json file "coordinates.json" indicating the protocol from the SKOS concept scheme (`cmp:https`), media type ("application/json"), JSONPath to extract data, access URL "https://ex.com/geodata/coordinates-.json", and fields that are going to be used in the transformation. There is no security scheme. The MySQL table "cities" also has no security scheme, the protocol needed is `cmp:jdbc`, the database access is specified in the endpoint URL, and the table as an SQL query. The fields are also specified, with the special case of "zipcodes" that needs a `cm:hasNestedFrame` to extract multiple values inside the field.

```
# Locations
:FrameLoc a cm:SourceFrame;
  cm:expression "$.coordinates[*]";
  cm:language ql:JSONPath ;
  cm:hasField :lat;
  cm:hasField :long;
  cm:hasField :loc_city;
  cm:hasDataSource [ a cm:SynchronousSource;
    dcat:mediaType "text/json";
    dcat:accessService [
      cm:hasProtocol cmp:https;
      dcat:endpointURL "https://ex.com/geodata/
          coordinates.json"
      cm:hasSecurityScheme [ a wotsec:NoSecurityScheme
          ; ];
    ] ;
  ] .

:lat a cm:DataField ; cm:field "$.latitude" .
:long a cm:DataField ; cm:field "$.longitude" .
:loc_city a cm:DataField; cm:field "$.city" .

# Cities
:FrameCities a cm:SourceFrame ;
  cm:expression "SELECT * FROM cities;";
  cm:hasField :c_city;
  cm:hasField :population;
  cm:hasField :year;
  cm:hasNestedFrame [
    cm:expression "$.zipcodes[*]";
    cm:hasField :zipcode ];
  cm:hasDataSource [ a cm:SynchronousSource;
    dcat:mediaType "text/plain";
    dcat:accessService [
      cm:hasProtocol cmp:jdbc;
      dcat:endpointURL "jdbc:mysql://localhost:3306/
          citydb";
      cm:hasSecurityScheme [a wotsec:NoSecurityScheme
          ;] ].

:c_city a cm:DataField; cm:field "city" .
:population a cm:DataField; cm:field "population" .
:year a cm:DataField; cm:field "year_modified" .
:zipcode a cm:DataField cm:field "zipcodes" .
```

Listing 8: Description with the Conceptual Mapping of two data sources (a JSON file and a relational database), their access and fields.

Statements. Listing 9 contains the rules needed to create instances of the classes `eg:Location` and `eg:City`; and their following attributes: `eg:lat` and `eg:long` for the former; `eg:zipcode` for the latter. To correctly generate the URI for the instances of `eg:City`, a replace function inside a concatenate function is needed to (1) remove the blank spaces in the field "city" and (2) add the field to the base URI "http://ex.com/city/".

```
# Locations
```

[13] https://saref.etsi.org/
[14] https://saref.etsi.org/core/v3.1.1/
[15] https://saref.etsi.org/extensions.html
[16] http://vocab.linkeddata.es/def/conceptual-mapping-csv

```
:SubjectLoc a cm:ReferenceNodeMap ;
  cm:hasEvaluableExpression [
    cm:hasFunctionName cmf:concat;
    cm:hasInput ([cm:constantValue "http://ex.com/loc/"]    :lat [cm:constantValue "-" ] :long)].

:StatementLoc1 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue rdf:type ] ];
  cm:object [cm:hasEvaluableExpression [cm:constantValue eg:Location]].

:StatementLoc2 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:lat]];
  cm:object [ a cm:Literal; cm:hasEvaluableExpression :lat];
  cm:hasDatatype [cm:hasEvaluableExpression xsd:decimal].

:StatementLoc3 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:long]];
  cm:object [ a cm:Literal; cm:hasEvaluableExpression :long];
  cm:hasDatatype [ cm:hasEvaluableExpression xsd:decimal].

# Cities
:city_ns a cm:FunctionExpression ;
  cm:functionName cmf:replace ;
  cm:hasInput (c_city " " "")

:SubjectCities a cm:ReferenceNodeMap;
  cm:hasEvaluableExpression [
    cm:hasFunctionName cmf:concat;
    cm:hasInput ([cm:constantValue "http://ex.com/city/"] :city_ns)].

:StatementCit1 a cm:StatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue rdf:type]];
  cm:object [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression  [cm:constantValue eg:City]] .

:StatementCit2 a cm:StatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue rdfs:label]];
  cm:object [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression  [cm:constantValue :c_city]] .
  cm:hasLanguage [ cm:hasEvaluableExpression [ cm:constantValue "es" ] ].

:StatementCit3 a cm:StatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:zipcode] ];
  cm:object [ a cm:Literal;
    cm:hasEvaluableExpression [cm:constantValue :zipcode] ];
  cm:hasDatatype [ cm:hasEvaluableExpression xsd:integer ].
```

Listing 9: Description with the Conceptual Mapping of the creation of regular statements from the data sources described in Listing 8.

Special statements. Listing 10 describes how a conditional statement and a linking rule are generated. This description is represented by means of functions. With the property `cm:hasBooleanCondition`, the conditional statement declares that the field `:year` has to be greater than 2020. The linking rule performs the link between the instances of `eg:City` and `eg:Location` with the predicate `eg:location`, using a distance metric (levenshtein function) that has to be greater then a threshold of "0.75".

```
:StatementCit4 a cm:ConditionalStatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:population] ];
  cm:object [ a cm:Literal;
    cm:hasEvaluableExpression  [cm:constantValue :population] ];
  cm:hasDatatype [ cm:hasEvaluableExpression xsd:integer];
  cm:hasBooleanCondition [
    cm:functionName cmf:greater_than ;
    cm:hasInput ( :year 2020 ) ] .

:LinkExp1 a cm:LinkingExpression ;
  cm:source :StatementCit1 ;
  cm:target :StatementLoc1 ;
  cm:property eg:location ;
  cm:hasBooleanCondition [
```

```
    cm:functionName cmf:greater_than ;
    cm:hasInput ( :levfun 0.75 ) ] .

:levfun a cm:FunctionExpression ;
  cm:functionName cmf:levenshtein_distance ;
  cm:hasInput (:c_city :loc_city) .
```

Listing 10: Conditional and linking rules described with the Conceptual Mapping that complement the data source description and regular statements described in Listing 8 and Listing 9.

6. Conceptual Mapping Publication and Maintenance

The ontology is considered ready for publication when it passes all evaluations. This means that it is correctly implemented in the formal language (OWL) and meets the requirements.

In order to publish the ontology, the first step required is to create the ontology documentation. We used Widoco [61], integrated inside the OnToology [62] system, to automatically generate and update the HTML documentation every time there is a commit in the GitHub repository where the ontology is stored. This documentation contains the ontology metadata, links to the previous version, a description of the ontology, the diagram, and detailed examples of the capabilities of the language. It is published using a W3ID URL[5] and under the CC BY-SA 4.0 license.

The HTML documentation is not the only documentation resource provided. An overview of all resources is provided in the ontology portal[3]. This portal shows in a table the ontologies associated with the Conceptual Mapping ontology. For now, the core (Conceptual Mapping) and an extension to describe CSV files in detail (Conceptual Mapping - CSV Description) are available. For each ontology, links to the HTML documentation, the requirements, the GitHub repository, the Issue Tracker, and the releases are provided.

The maintenance is supported by the Issue Tracker[17], where proposals for new requirements, additions, deletions or modifications can be added as GitHub issues. This approach allows authors to review the proposals and discuss their possible implementation.

[17] https://github.com/oeg-upm/Conceptual-Mapping/issues

7. Conclusion and Future Work

This paper presents the Conceptual Mapping, an ontology-based mapping language that aims to gather the expressiveness of current declarative mapping languages. In order to build this ontology, we first conducted an extensive analysis of the state-of-the-art mapping specifications (presented as a comparison framework) and mapping challenges proposed by the community, improving the understanding of current mapping languages and expanding previous studies on the comparison of language characteristics. Then, this analysis allowed us to develop a unique model that aims to integrate the common features of existing languages, acknowledging the limitations of representing the full potential of SPARQL-based languages such as SPARQL-Generate or Facade-X. Next, the approach was evaluated by validating that the constructs provided by this language can address the requirements extracted from the two-fold analysis. Thus, we ensure that this language covers the required expressiveness. The language is formalized as an ontology that is available along with a documentation online.

Our future work lines include exploring the limitations of the current scope and addressing the gap to be able to represent the expressiveness of SPARQL-based languages. Similarly to a programming language, SPARQL-based languages can specify "instructions" to describe and transform data that is not accessible by other languages, because of inner restrictions or simply because they lack the necessary constructs. At some point, modelling constructs for each specific use case becomes unfeasible, unpractical and very likely, too verbose. Despite the difficulties, we want to keep updating with modules our ontology with new issues and addressing the limitations to a reasonable extent. We also want to explore the possibility of implementing this ontology as a common interchange language for mapping translation purposes [65, 66] that we believe can help build bridges toward mapping interoperability. We also consider the integration of the mapping translation step into the common workflow for constructing virtual and materialized Knowledge Graphs, using this conceptual model as the core resource for carrying out this process. Furthermore, we want to integrate this ontology into previous work on mapping rules management, MappingPedia [67], with the translation step between different specifications. In this manner, we aim to help users and practitioners during the selection of mapping languages and engines, not forcing them to select the ones that are only under their control, but being able to select the ones that best fit their own specific use cases. Finally, we want to specify the correspondence of concepts between the considered mapping languages and the Conceptual Mapping, and to formally define the semantics and operators required to perform the mapping translation, adapting previous works on schema and data translations [68, 69].

Acknowledgements

We are thankful for the feedback provided by Anastasia Dimou during the elaboration of this paper. The work presented in this paper is supported by the Spanish Ministerio de Ciencia e Innovación funds under the Spanish I+D+I national project KnowledgeSpaces: Técnicas y herramientas para la gestión de grafos de conocimientos para dar soporte a espacios de datos (PID2020-118274RB-I). Also, this work is partially funded by the EuropeanUnion's Horizon 2020 Research and Innovation Programme through the AURORAL project, Grant Agreement No. 101016854. David Chaves-Fraga is supported by the Spanish Minister of Universities (Ministerio de Universidades) and by NextGenerationEU funds through the Margarita Salas postdoctoral fellowship.

References

[1] 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.

[2] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G.D. Melo, C. Gutierrez, S. Kirrane, J.E. Labra-Gayo, R. Navigli, S. Neumaier et al., Knowledge graphs, *ACM Computing Surveys (CSUR)* **54**(4) (2021), 1–37. https://doi.org/10.1145/3447772.

[3] F. Michel, J. Montagnat and C.F. Zucker, A survey of RDB to RDF translation approaches and tools, Technical Report, 2014. https://hal.archives-ouvertes.fr/hal-00903568.

[4] J. Arenas-Guerrero, M. Scrocca, A. Iglesias-Molina, J. Toledo, L.P. Gilo, D. Dona, O. Corcho and D. Chaves-Fraga, Knowledge Graph Construction with R2RML and RML: An ETL System-based Overview, in: *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Online*, CEUR Workshop Proceedings, Vol. 2873, CEUR-WS.org, 2021. http://ceur-ws.org/Vol-2873/paper11.pdf.

[5] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini and R. Rosati, Linking Data to Ontologies, in: *Journal on Data Semantics X*, S. Spaccapietra, ed., Springer, Berlin, Heidelberg, 2008, pp. 133–173. doi:10.1007/978-3-540-77688-8_5.

[6] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati and M. Zakharyaschev, Ontology-Based Data Access: A Survey, in: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, Stockholm, Sweden*, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 5511–5519. doi:10.24963/ijcai.2018/777.

[7] A. Chebotko, S. Lu and F. Fotouhi, Semantics preserving SPARQL-to-SQL translation, *Data & Knowledge Engineering* **68**(10) (2009), 973–1000. doi:10.1016/j.datak.2009.04.001.

[8] A. Dimou, M.V. Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Van De Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: *Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (LDOW@WWW 2014), Seoul, Korea*, CEUR Workshop Proceedings, Vol. 1184, CEUR-WS.org, 2014. http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[9] 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), Lisbon, Portugal*, SciTePress, 2015, pp. 443–454, INSTICC. doi:10.5220/0005448304430454.

[10] H. García-González, I. Boneva, S. Staworko, J.E. Labra-Gayo and J.M. Cueva-Lovelle, ShExML: improving the usability of heterogeneous data mapping languages for first-time users, *PeerJ Computer Science* **6** (2020), e318. doi:10.7717/peerj-cs.318.

[11] S. Das, S. Sundara and R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012, *www.w3.org/TR/r2rml* (2012).

[12] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: *Proceedings of the 14th European Semantic Web Conference, Portorož, Slovenia*, Springer, Cham, 2017, pp. 35–50. doi:10.1007/978-3-319-58068-5_3.

[13] 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). doi:10.3390/s20030822.

[14] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi and K. Kajimoto, Web of Things (WoT) Architecture, W3C Recommendation 9 April 2020, *https://www.w3.org/TR/wot-architecture/* (2020).

[15] S. Kaebisch, T. Kamiya, M. McCool, V. Charpenay and M. Kovatsch, Web of Things (WoT) Thing Description. W3C Recommendation 9 April 2020., *https://www.w3.org/TR/wot-thing-description/* (2020).

[16] F. Priyatna, R. Alonso-Calvo, S. Paraiso-Medina and O. Corcho, Querying clinical data in HL7 RIM based relational model with morph-RDB, *Journal of biomedical semantics* **8**(1) (2017), 1–12. doi:10.1186/s13326-017-0155-8.

[17] B. De Meester, W. Maroy, A. Dimou, R. Verborgh and E. Mannens, Declarative data transformations for linked data generation: The case of DBpedia, in: *Proceedings*

of the 14th European Semantic Web Conference, Portorož, Slovenia, Vol. 10250, Springer, Cham, 2017, pp. 33–48. doi:10.1007/978-3-319-58451-5_3.

[18] K. Kyzirakos, D. Savva, I. Vlachopoulos, A. Vasileiou, N. Karalis, M. Koubarakis and S. Manegold, GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings, *Journal of Web Semantics* **52-53** (2018), 16–32, Elsevier. doi:10.1016/j.websem.2018.08.003.

[19] F. Michel, F. Gandon, V. Ah-Kane, A. Bobasheva, E. Cabrio, O. Corby, R. Gazzotti, A. Giboin, S. Marro, T. Mayer et al., Covid-on-the-Web: Knowledge graph and services to advance COVID-19 research, in: *Proceedings of the 19th International Semantic Web Conference, Athens, Greece*, Springer, Cham, 2020, pp. 294–310. doi:10.1007/978-3-030-62466-8_19.

[20] 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, Elsevier. doi:10.1016/j.websem.2020.100596.

[21] B. De Meester, P. Heyvaert, R. Verborgh and A. Dimou, Mapping languages analysis of comparative characteristics, in: *1st International Workshop on Knowledge Graph Building and Large Scale RDF Analytics, co-located with the 16th Extended Semantic Web Conference (ESWC 2019), Portorož, Slovenia*, CEUR Workshop Proceedings, Vol. 2489, CEUR-WS.org, 2019. http://ceur-ws.org/Vol-2489/paper4.pdf.

[22] A. Iglesias-Molina, D. Chaves-Fraga, F. Priyatna and O. Corcho, Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings., in: *12th International Conference on Semantic Web Applications and Tools for Health Care and Life Sciences, Edinburgh, Scotland, UK*, CEUR Workshop Proceedings, Vol. 2849, CEUR-WS.org, 2019. http://ceur-ws.org/Vol-2849/paper-01.pdf.

[23] M. Poveda-Villalón, A. Fernández-Izquierdo, M. Fernández-López and R. García-Castro, LOT: An industrial oriented ontology engineering framework, *Engineering Applications of Artificial Intelligence* **111** (2022), 104755, Elsevier. doi:10.1016/j.engappai.2022.104755.

[24] R. Albertoni, D. Browning, S. Cox, A. González Beltrán, A. Perego, P. Winstanley, F. Maali and J. Erickson, Data Catalog Vocabulary (DCAT), W3C Recommendation 04 February 2020, *https://www.w3.org/TR/vocab-dcat-2/* (2020).

[25] C. Bizer and A. Seaborne, D2RQ-treating non-RDF databases as virtual RDF graphs, in: *3rd International Semantic Web Conference (ISWC2004) Posters, Hiroshima, Japan*, 2004. http://iswc2004.semanticweb.org/posters/PID-SMCVRKBT-1089637165.pdf.

[26] R. Cyganiak, C. Bizer, J. Garbers, O. Maresch and C. Becker, The D2RQ Mapping Language, 2012. http://d2rq.org/d2rq-language.

[27] J. Barrasa, Ó. Corcho and A. Gómez-Pérez, R2O, an extensible and semantically based database-to-ontology mapping language, in: *Proceedings of the 2nd Workshop on Semantic Web and Databases, Toronto, Canada*, Vol. 14, Springer-Verlag, 2004, pp. 1069–1070. doi:10.1007/b106149.

[28] F. Michel, L. Djimenou, C.F. Zucker and J. Montagnat, xR2RML: Relational and Non-Relational Databases to RDF Mapping Language, 2017. https://www.i3s.unice.fr/~fmichel/xr2rml_specification_v5.html.

[29] A. Dimou, M. Vander Sande, B. De Meester, P. Heyvaert and T. Delva, RDF Mapping Language (RML), 2020. https://rml.io/specs/rml/.

[30] J. Slepicka, C. Yin, P.A. Szekely and C.A. Knoblock, KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources., in: *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, US*, CEUR Workshop Proceedings, Vol. 1426, CEUR-WS.org, 2015. http://ceur-ws.org/Vol-1426/paper-08.pdf.

[31] A.C. Junior, C. Debruyne, R. Brennan and D. O'Sullivan, FunUL: a method to incorporate functions into uplift mapping languages, in: *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, Singapore, Singapore*, ACM New York, NY, USA, 2016, pp. 267–275. doi:10.1145/3011141.3011152.

[32] C. Debruyne and D. O'Sullivan, R2RML-F: towards sharing and executing domain logic in R2RML mappings, in: *Workshop on Linked Data on the Web co-located with 25th International World Wide Web Conference (LDOW@WWW 2016), Florence, Italy*, CEUR Workshop Proceedings, Vol. 1593, CEUR-WS.org, 2016. http://ceur-ws.org/Vol-1593/article-13.pdf.

[33] A. Chortaras and G. Stamou, D2RML: Integrating Heterogeneous Data and Web Services into Custom RDF Graphs., in: *Workshop on Linked Data on the Web co-located with The Web Conference 2018 (LDOW@WWW 2018), Lyon, France*, CEUR Workshop Proceedings, Vol. 2073, CEUR-WS.org, 2018. http://ceur-ws.org/Vol-2073/article-07.pdf.

[34] A. Langegger and W. Wöß, XLWrap–querying and integrating arbitrary spreadsheets with SPARQL, in: *Proceedings of the 8th International Semantic Web Conference (ISWC 2009) Chantilly, VA, USA*, Vol. 5823, Springer, Berlin, Heidelberg, 2009, pp. 359–374. doi:10.1007/978-3-642-04930-9_23.

[35] A. Langegger, XLWrap – Spreadsheet-to-RDF Wrapper, 2009. https://xlwrap.sourceforge.io/.

[36] J. Tennison, G. Kellogg and I. Herman, Model for tabular data and metadata on the web, W3C Recommendation 17 December 2015, *https://www.w3.org/TR/tabular-data-model/* (2015).

[37] M. Lefrançois, A. Zimmermann, N. Bakerally, E.M. Khalfi and O. Qawasmeh, SPARQL-Generate - query and generate both RDF and text, 2022. https://ci.mines-stetienne.fr/sparql-generate/index.html.

[38] S. Bischof, S. Decker, T. Krennwallner, N. Lopes and A. Polleres, Mapping between RDF and XML with XSPARQL, *Journal on Data Semantics* **1**(3) (2012), 147–185. doi:10.1007/s13740-012-0008-7.

[39] A. Polleres, T. Krennwallner, N. Lopes, J. Kopecky and S. Decker, XSPARQL Language Specification, 2009. https://www.w3.org/Submission/xsparql-language-specification/.

[40] TARQL: SPARQL for Tables, 2019. http://tarql.github.io/.

[41] E. Daga, L. Asprino, P. Mulholland and A. Gangemi, Facade-X: An Opinionated Approach to SPARQL Anything, in: *Volume 53: Further with Knowledge Graphs*, Vol. 53, M. Alam, P. Groth, V. de Boer, T. Pellegrini and H.J. Pandit, eds, IOS Press, 2021, pp. 58–73. doi:10.3233/ssw210035.

[42] SPARQL Anything, 2022. https://sparql-anything.readthedocs.io/en/latest/.

[43] S. Union, SMS2 (Stardog Mapping Syntax 2), 2021. https://docs.stardog.com/archive/7.5.0/virtual-graphs/mapping-data-sources.html.

[44] A. Cimmino and R. García-Castro, Helio Mappings, 2020. https://github.com/oeg-upm/helio/wiki/Helio-Materialiser-for-Users#helio-mappings.

[45] B. Vu, J. Pujara and C.A. Knoblock, D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF, in: *Proceedings of the 10th International Conference on Knowledge Capture (K-CAP 2019), Marina del Rey, CA, USA*, ACM New York, NY, USA, 2019, pp. 189–196. doi:10.1145/3360901.3364449.

[46] H. García-González, A ShExML perspective on mapping challenges: already solved ones, language modifications and future required actions, in: *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Online*, CEUR Workshop Proceedings, Vol. 2873, CEUR-WS.org, 2021, pp. 1–14. http://ceur-ws.org/Vol-2873/paper2.pdf.

[47] H. García-González, Shape Expressions Mapping Language (ShExML), 2022. http://shexml.herminiogarcia.com/spec/.

[48] Z. GmbH, Expressive RDF Mapper (XRM), 2022. https://zazuko.com/products/expressive-rdf-mapper/.

[49] C. Stadler, J. Unbehauen, P. Westphal, M.A. Sherif and J. Lehmann, Simplified RDB2RDF mapping, in: *Workshop on Linked Data on the Web co-located with the 24th International World Wide Web Conference (LDOW@WWW 2015), Florence, Italy*, CEUR Workshop Proceedings, Vol. 1409, CEUR-WS.org, 2015. http://ceur-ws.org/Vol-1409/article-07.pdf.

[50] M. Rodriguez-Muro and M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, *Journal of Web Semantics* **33** (2015), 141–169. doi:10.1016/j.websem.2015.03.001.

[51] P. Heyvaert, B. De Meester, A. Dimou and R. Verborgh, Declarative Rules for Linked Data Generation at your Fingertips!, in: *The Semantic Web: ESWC 2018 Satellite Events, Crete, Greece*, Vol. 11155, Springer, Cham, 2018, pp. 213–217. doi:10.1007/978-3-319-98192-5_40.

[52] S. Harris, A. Seaborne and E. Prud'hommeaux, SPARQL 1.1, Query Language, W3C Recommendation 21 March 2013, *https://www.w3.org/TR/sparql11-query/* (2013).

[53] E. Prud'hommeaux, J.E. Labra Gayo and H. Solbrig, Shape expressions: an RDF validation and transformation language, in: *Proceedings of the 10th International Conference on Semantic Systems (SEMANTiCS 2014), Leipzig, Germany*, Association for Computing Machinery, 2014, pp. 32–40. doi:10.1145/2660517.2660523.

[54] 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 (HAIS 2018), Oviedo, Spain*, Vol. 10870, Springer, Cham, 2018, pp. 145–157. doi:10.1007/978-3-319-92639-1_13.

[55] M. Hert, G. Reif and H.C. Gall, A comparison of RDB-to-RDF mapping languages, in: *Proceedings of the 7th International Conference on Semantic Systems, Graz, Austria*, ACM New York, NY, USA, 2011, pp. 25–32. doi:10.1145/2063518.2063522.

[56] M.C. Suárez-Figueroa, A. Gómez-Pérez and M. Fernandez-Lopez, The NeOn Methodology framework: A scenario-based methodology for ontology development, *Applied ontology* **10**(2) (2015), 107–145. doi:10.3233/AO-150145.

[57] 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), Tangier, Morocco*, IEEE, 2021, pp. 1–8. doi:10.1109/AICCSA53542.2021.9686935.

[58] S. Chávez-Feria, R. García-Castro and M. Poveda-Villalón, Chowlk: from UML-based ontology conceptualizations to OWL, in: *The Semantic Web: 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece*, Springer, Cham, 2022, pp. 338–352. doi:10.1007/978-3-031-06981-9_20.

[59] M. Poveda-Villalón, A. Gómez-Pérez and M.C. Suárez-Figueroa, Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation, *International Journal on Semantic Web and Information Systems (IJSWIS)* **10**(2) (2014), 7–34. doi:10.4018/ijswis.2014040102.

[60] D. Garijo, O. Corcho and M. Poveda-Villalón, FOOPS!: An Ontology Pitfall Scanner for the FAIR principles, in: *International Semantic Web Conference (ISWC) 2021: Posters, Demos, and Industry Tracks*, CEUR Workshop Proceedings, Vol. 2980, CEUR-WS.org, 2021. http://ceur-ws.org/Vol-2980/paper321.pdf.

[61] D. Garijo, WIDOCO: a wizard for documenting ontologies, in: *6th International Semantic Web Conference, Vienna, Austria*, Springer, Cham, 2017, pp. 94–102. doi:10.1007/978-3-319-68204-4_9.

[62] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Perez, A. Fernández-Izquierdo and O. Corcho, Automating ontology engineering support activities with OnToology, *Journal of Web Semantics* **57** (2019), 100472. doi:10.1016/j.websem.2018.09.003.

[63] J. Pérez, M. Arenas and C. Gutierrez, Semantics and complexity of SPARQL, *ACM Transactions on Database Systems (TODS)* **34**(3) (2009), 1–45. doi:10.1145/1567274.1567278.

[64] T. Delva, D. Van Assche, P. Heyvaert, B. De Meester and A. Dimou, Integrating nested data into knowledge graphs with RML fields, in: *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Online*, CEUR Workshop Proceedings, Vol. 2873, CEUR-WS.org, 2021, pp. 1–16. http://ceur-ws.org/Vol-2873/paper9.pdf.

[65] O. 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.

[66] A. Iglesias-Molina, A. Cimmino and O. Corcho, Devising Mapping Interoperability with Mapping Translation, in: *Proceedings of the 3rd International Workshop on Knowledge Graph Construction co-located with 19th Extended Semantic Web Conference (ESWC 2022), Hersonissos, Greece*, CEUR Workshop Proceedings, Vol. 3141, CEUR-WS.org, 2022, pp. 1–8. http://ceur-ws.org/Vol-3141/paper6.pdf.

[67] F. Priyatna, E. Ruckhaus, N. Mihindukulasooriya, Ó. Corcho and N. Saturno, Mappingpedia: A collaborative environment for R2RML mappings, in: *ESWC 2017: The Semantic Web: ESWC 2017 Satellite Events, Portorož, Slovenia*, Springer, Cham, 2017, pp. 114–119. doi:10.1007/978-3-319-70407-4_22.

[68] M. Arenas, J. Pérez, J.L. Reutter and C. Riveros, Foundations of schema mapping management, in: *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, Indianapolis,*

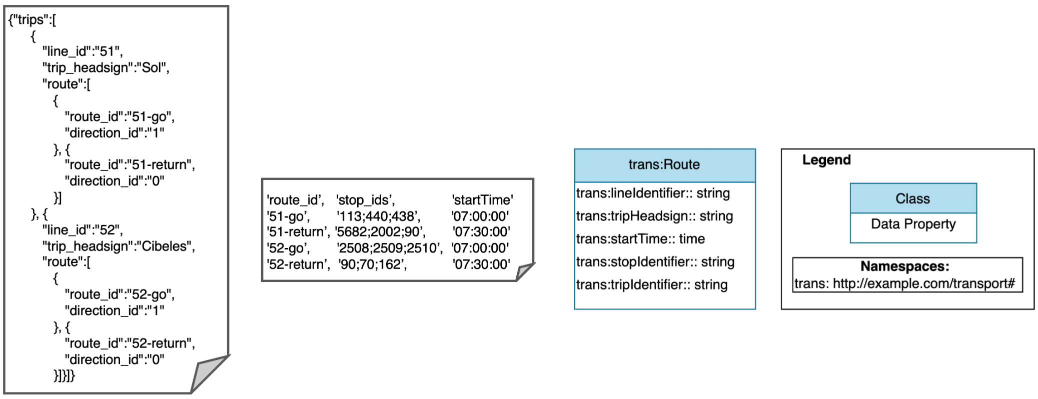
USA, ACM New York, NY, USA, 2010, pp. 227–238. doi:10.1145/1807085.1807116.
[69] M. Arenas, J. Pérez and C. Riveros, The recovery of a schema mapping: bringing exchanged data back, *ACM Transactions on Database Systems (TODS)* **34**(4) (2009), 1–48. doi:10.1145/1620585.1620589.

Appendix A. Example - Routes

The following example (Listing 11) illustrates features of the ontology that do not appear in the example shown in Section 5.5. Together, both examples shows the core features of the ontology, further possibilities can be achieved by combining the shown constructs. This example shows how to describe a JSON file, "trips.json"(Fig. 6a) and CSV file (Fig. 6b) following the ontology described by Fig. 6c. This ontology is composed of one class `trans:Route`. The routes are described with the properties `trans:lineIdentifier`, `trans:tripHeadsign`, `trans:startTime`, `trans:stopIdentifier` and `trans:tripIdentifier`. The fields created contain information from different levels of iteration from the JSON file and fields from the CSV file.

The mapping presented joins two sources (`:FrameRouteStop`). It uses a `CombinedFrame`, that joins two `SourceFrame`, one that describes a json file, "trips.json" (`:FrameRoute`), and another that describes a csv file, "route_stop.csv" (`:FrameStop`). The join is performed by joining the fields `:s_route_id` and `:r_route_id`. The JSON file is retrieved from an API using `wotsec:APIKeySecurityScheme`, and is retrieved asynchronously every 300000 ms (5 minutes).

Finally, the mapping rules create the values from the data properties of the class `trans:Route` from two different sources joined as one frame, separately or in one single object, like `:StatementRoute6`. Additionally, `:StatementRoute5` creates a list of values for the stops ids using a split function to separate the original value.



(a) "trips.json" (b) "route_stop.csv" (c) Example reference ontology for the class Route

Fig. 6. Data sources (a,b) and reference ontology used for the example represented in Listing 11.

```
:FrameRoute a cm:SourceFrame ;
   cm:expression "$.trips[*]" ;
   cm:language ql:JSONPath ;
   cm:hasField :line_id ;
   cm:hasField :trip_headsign ;
   cm:hasNestedFrame [
      cm:expression "route[*]" ;
      cm:hasField :r_route_id ;
      cm:hasField :direction_id ] ;
   cm:hasDataSource [ a cm:SynchronousSource ;
      cm:encoding "utf-8" ;
      dcat:mediaType "application/json" ;
      dcat:accessService [
         cm:hasProtocol cmp:https ;
         dcat:endpointURL <https://ex.com/transport/trips.json> ;
         cm:hasSecurityScheme [ a wotsec:APIKeySecurityScheme ;
            wotsec:in "header" ;
            wotsec:name "api_key" ]; ] ; ] .
```

```

:FrameStop a cm:SourceFrame ;
    cm:language ql:CSV ;
    cm:hasField :s_route_id ;
    cm:hasField :stops_ids ;
    cm:hasField :start_time ;
    cm:hasDataSource [ a cm:SynchronousSource ;
        dcat:mediaType "text/csv" ;
        dcat:accessService [
            cm:hasProtocol cmp:file ;
            dcat:endpointURL "file:///user/data/route_stop.csv" ;
            cm:hasSecurityScheme [ a wotsec:NoSecurityScheme ; ]; ] ; ] .

:FrameRouteStop a cm:JoinCombination ;
  cm:combinesFrame :FrameStop ;
  cm:combinesFrame :FrameRoute ;
  cm:joinsBy :s_route_id ;
  cm:joinsBy :r_route_id .

:s_route_id a cm:DataField; cm:field "route_id" .
:stops_ids a cm:DataField; cm:field "stops_ids" .
:start_time a cm:DataField; cm:field "start_time" .
:line_id a cm:DataField; cm:field "line_id" .
:trip_headsign a cm:DataField; cm:field "trip_headsign" .
:r_route_id a cm:DataField; cm:field "route_id" .
:direction_id a cm:DataField; cm:field "direction_id" .

# Rules
:SubjectRoute a cm:ReferenceNodeMap ;
    cm:hasEvaluableExpression [
        cm:hasFunctionName cmf:concat;
        cm:hasInput ([cm:constantValue "http://ex.com/route/"] :s_route_id ] .

:StatementRoute1 a cm:StatementMap ;
    cm:hasFrame :FrameRouteStop ;
    cm:subject :SubjectRoute ;
    cm:predicate [ a cm:ReferenceNodeMap;
        cm:hasEvaluableExpression [cm:constantValue rdf:type ] ];
    cm:object [ cm:hasEvaluableExpression [cm:constantValue trans:Route] ].

:StatementRoute2 a cm:StatementMap ;
    cm:hasFrame :FrameRouteStop ;
    cm:subject :SubjectRoute ;
    cm:predicate [ a cm:ReferenceNodeMap;
        cm:hasEvaluableExpression [cm:constantValue trans:lineIdentifier ] ];
    cm:object [ cm:hasEvaluableExpression [cm:constantValue :line_id] ].

:StatementRoute3 a cm:StatementMap ;
    cm:hasFrame :FrameRouteStop ;
    cm:subject :SubjectRoute ;
    cm:predicate [ a cm:ReferenceNodeMap;
        cm:hasEvaluableExpression [cm:constantValue trans:tripHeadsign ] ];
    cm:object [ cm:hasEvaluableExpression [cm:constantValue :trip_headsign] ];
    cm:language [ cm:hasEvaluableExpression [cm:constantValue "es"] ].

:StatementRoute4 a cm:StatementMap ;
    cm:hasFrame :FrameRouteStop ;
    cm:subject :SubjectRoute ;
    cm:predicate [ a cm:ReferenceNodeMap;
        cm:hasEvaluableExpression [cm:constantValue trans:startTime ] ];
    cm:object [ cm:hasEvaluableExpression [cm:constantValue :start_time] ];
```

```
    cm:datatype [ cm:hasEvaluableExpression [cm:constantValue xsd:time] ].

:StatementRoute5 a cm:StatementMap ;
    cm:hasFrame :FrameRouteStop ;
    cm:subject :SubjectRoute ;
    cm:predicate [ a cm:ReferenceNodeMap;
        cm:hasEvaluableExpression [cm:constantValue trans:stopIdentifier ] ];
    cm:object [ a cm:ListMap; cm:hasEvaluableExpression :city_ns ].

:city_ns a cm:FunctionExpression ;
    cm:functionName cmf:split ;
    cm:hasInput (:stop_ids ";") .

:StatementRoute6 a cm:StatementMap ;
    cm:hasFrame :FrameRouteStop ;
    cm:subject :SubjectRoute ;
    cm:predicate [ a cm:ReferenceNodeMap;
        cm:hasEvaluableExpression [cm:constantValue trans:tripIdentifier ] ];
    cm:object [ cm:hasEvaluableExpression :trip_id ].

:trip_id a FunctionExpression;
    cm:hasFunctionName cmf:concat;
    cm:hasInput (:s_route_id [cm:constantValue "_"]
                 :direction_id [ cm:constantValue "_" ] :start_time) .
```

Listing 11: Routes mapping example, uses as input the data sources and ontology in Fig. 6 and outputs Listing 12.

```
<http://ex.com/route/51-go> a trans:Route ;
  trans:lineIdentifier "51" ;
  trans:tripHeadsign "Sol"@es ;
  trans:startTime "07:00:00"^^xsd:time ;
  trans:stopIdentifier ("113" "440" "438") ;
  trans:tripIdentifier "51-go\_1\_07:00:00" .

<http://ex.com/route/51-return> a trans:Route ;
  trans:lineIdentifier "51" ;
  trans:tripHeadsign "Sol"\@es ;
  trans:startTime "07:30:00"^^xsd:time ;
  trans:stopIdentifier ("5682" "2002" "90") ;
  trans:tripIdentifier "51-go\_0\_07:30:00" .

<http://ex.com/route/52-go> a trans:Route ;
  trans:lineIdentifier "52" ;
  trans:tripHeadsign "Cibeles"\@es ;
  trans:startTime "07:00:00"^^xsd:time ;
  trans:stopIdentifier ("2508" "2509" "2510") ;
  trans:tripIdentifier "52-go\_1\_07:00:00" .

<http://ex.com/route/52-return> a trans:Route ;
  trans:lineIdentifier "52" ;
  trans:tripHeadsign "Cibeles"\@es ;
  trans:startTime "07:30:00"^^xsd:time ;
  trans:stopIdentifier ("90" "70" "162") ;
  trans:tripIdentifier "52-go\_0\_07:30:00" .
```

Listing 12: Result from Routes mapping represented in Listing 11.

Appendix B. Framework Comparison of Existing Mapping Languages

Table 2

Data retrieval and data source expression for the analysed mapping languages from the references stated in Table 1. (*) indicates features not explicitly declared in the language, but that are implemented by compliant tools.

Feature \Language		ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FunUL	Helio	WoT	D-REPR	XLWrap	D2RML	SPARQL-Anything	SMS2
Retrieval of data	Streams	false	false	false	false	false	true*[a]	false	false	true	false	false	false	false	false	false	false	true	false
	Synchronous sources	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true
	Asynchronous sources	-	-	-	-	-	-	-	-	Events, Periodic	-	-	Periodic	-	-	-	-	-	-
Expressing data sources	Security terms	-	-	-	-	Basic (DB)	Basic (DB)	Basic (DB)	Basic (DB)	-	Basic (DB)	-	API Key, OAuth2, Bearer, Basic	API Key, OAuth2, Bearer, Basic	-	-	Basic (DB)	-	Basic (DB)
	Encoding	false	false	true*[b]	true	false	true	false	false	false	false	false	true	true	false	false	false	true	
	MIME Type	false	false	false	false	false	false	false	false	false	false	false	true	true	false	false	false	true	false
	Features describing data	Iterator, Queries	-	Delimiter, Separator	Delimiter, Separator, Regex	Queries	Delimiter, Regex, Iterator, Queries, Separator	Queries	Regex, Iterator, Queries	Delimiter, Regex, Iterator, Queries, Separator	Iterator, Queries	Iterator, Queries	Delimiter, Regex, Iterator, Queries, Separator	Iterator	Delimiter, Regex, Iterator	Separator	Delimiter, Regex, Iterator, Queries	Delimiter, Regex, Iterator, Queries, Separator	Delimiter, Regex, Iterator, Queries, Separator
	Retrieval protocol	file, http(s), odbc/jdbc	file	file	file, http(s)	file, http(s), odbc/jdbc	file, http(s), odbc/jdbc	file, odbc/jdbc	file, odbc/jdbc	file, http(s), odbc/jdbc WebSocket, MQTT	file, http(s), odbc/jdbc	file, http(s)	file, any URI-based	http(s)	file	file	file, http(s), odbc/jdbc	file, http(s)	file, odbc/jdbc
	Data formats	Tabular, Tree, Graph	Tree (XML)	Tabular (CSV)	Tabular	Tabular	Tabular, Tree, Graph	Tabular, Tree	Tabular, Tree	Tabular, Tree, Plain Text, Graph	Tabular	Tabular, Graph	Tabular, Tree, Plain Text, Graph	Tree (JSON)	Tabular (CSV), Tree	Tabular (CSV, Excel)	Tabular, Tree, Plain Text, Graph	Tabular, Tree, Plain Text, Graph	Tabular, Tree, Plain Text, Graph

[a] Implemented by RMLSreamer, available at https://github.com/RMLio/RMLStreamer.
[b] Command line input option `--encoding` [40].

Table 3: Features for subject, predicate, and object generation of the studied mapping languages from the references stated in Table 1.

Feature & Language			ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FunUL	Helio	WoT	D-REPR	XLWrap	D2RML	SPARQL-Anything	SMS2
Subject	Constant		IRI	BN, IRI	BN, IRI	IRI	BN, IRI	BN, IRI	-	BN, IRI	BN, IRI	BN, IRI	BN, IRI	IRI	IRI	BN, IRI	BN, IRI	BN, IRI	BN, IRI	BN, IRI
	Dynamic	RDF Resource	IRI	BN, IRI	BN, IRI	IRI	BN, IRI	BN, IRI	IRI	BN, IRI	IRI	BN, IRI	BN, IRI	IRI	-	BN, IRI	BN, IRI	BN, IRI	IRI	BN, IRI
		Data Reference	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	-	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format
		Data Sources	1..*	1..*	1..1	1..1	1..1	1..1	1..*	1..*	1..*	1..1	1..1	1..*	-	1..1	1..1	1..1	1..*	1..1
		Hierarchy Iteration	true	true	false	false	false	true	true	false	true	false	false	false	false	true	false	true	true	true
		Functions	-	1..*	1..*	-	-	1..*	1..*	-	1..*	1..*	1..*	1..*	-	1..*	1..*	1..*	1..*	1..*
Predicate	Constant		IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	
	Dynamic	RDF Resource	-	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	-	IRI	-	IRI	IRI	IRI
		Data Reference	-	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	-	1..* Ref 1..1 Format	-	1..* Ref 1..1 Format	1..* Ref 1..* Format	-
		Data Sources	-	1..1	1..1	1..1	1..1	1..1	1..*	1..1	1..*	1..1	1..1	1..*	-	1..1	-	1..1	1..*	-
		Hierarchy Iteration	false	false	false	false	false	true	true	false	false	false	false	false	false	true	false	true	true	false
		Functions	-	1..*	1..*	-	-	1..*	1..*	-	1..*	1..*	1..*	1..*	-	1..*	-	1..*	1..*	-
Object	Constant		IRI, Literal	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal	IRI, Literal	IRI, Literal	IRI, Literal	BN, IRI, Literal, List, Container	BN, IRI, Literal, List	IRI, Literal	IRI, Literal	IRI, Literal	BN, IRI, Literal, List, Container	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal, List	BN, IRI, Literal
	Dynamic	RDF Resource	IRI, Literal, Lists	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal, List	BN, IRI, Literal, List, Container	BN, IRI, Literal, List	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal	IRI, Literal	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal, List	BN, IRI, Literal
		Data Reference	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	-	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format
		Data Sources	1..*	1..*	1..1	1..1	1..1	1..1	1..*	1..*	1..*	1..1	1..1	1..*	-	1..1	1..1	1..1	1..*	1..1
		Hierarchy Iteration	true	true	false	false	false	true	true	false	true	false	false	false	false	true	false	true	true	true
		Functions	1	1..*	1..*	-	-	1..*	1..*	-	1..*	1..*	1..*	1..*	-	1..*	1..*	1..*	1..*	1..*
	Datatype and Language		static, dynamic	static, dynamic	static, dynamic	static	static	static, dynamic	-	static	static	static	static	static, dynamic	static	static	-	static	static	static

Table 4: Statements, linking rules, and function properties of the studied mapping languages from the references stated in Table 1.

Feature \Language		ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FunUL	Helio	WoT	D-REPR	XLWrap	D2RML	SPARQL-Anything	SMS2
Statements	Assign to named graphs	static	-	-	-	static, dynamic	static, dynamic	static	static	-	static, dynamic	static, dynamic	-	-	-	static	static, dynamic	-	-
	Retrieve data from one source	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true	true
	Retrieve data from one or more sources	true	true	false	true	false	false	false	false	true	false	false	true	false	true	false	true	true	false
	Allow conditions to form statements	true	true	true	false	false	false	false	false	true	false	false	false	false	false	true	true	true	false
Linking rules	Use one data reference	true	true	false	true	true	true	false	true	true	true	true	true	true	true	false	true	true	false
	Use one or more data reference	true	false	false	false	false	true	false	true	true	false	false	true	false	true	false	false	true	false
	No condition to link	true	true	false	false	true	true	false	true	true	true	true	true	false	false	false	true	true	false
	Link with one condition	true	true	false	false	true	true	false	true	true	true	true	true	false	true	false	true	true	false
	Link with one or more conditions	false	true	false	false	true	true	false	true	true	true	true	true	false	true	false	true	true	false
	Use only equal function in condition	true	true	false	false	true	true	false	true	true	true	true	true	false	true	false	true	true	false
	Use any similarity function in condition	false	true	false	false	false	true	false	false	false	false	false	true	false	false	false	true	true	false
Functions	Cardinality	1:1, N:1	1:1, N:1, 1:N, N:M	1:1, N:1	-	-	1:1, N:1*[a]	1:1, N:1, 1:N, N:M	-	1:1, N:1, 1:N, N:M	1:1, N:1	1:1, N:1	1:1, N:1	-	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M
	Nested functions	false	true	true	false	false	true*[a]	false	false	true	true	true	true	false	true	true	true	true	true
	Functions belong to a specification	true	false	true	false	false	true*[a]	false	false	true	false	false	true	false	false	true	false	true	true
	Declare own functions	true	true	false	false	false	true*[a]	false	false	true	true	true	false	false	false	true	true	true	false

[a] With the Function Ontology (FnO) [17]