Towards an Ontological Approach for Integrating Declarative Mapping Languages

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

[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
E-mails: ana.iglesiasm@upm.es, cimmino@fi.upm.es, eruckhaus@fi.upm.es, dchaves@fi.upm.es, rgarcia@fi.upm.es, ocorcho@fi.upm.es

Abstract. Nowadays, Knowledge Graphs are extensively created using very different techniques: tools such as OpenRefine, programming scripts or mapping languages, among them. Focusing on the latter, the wide variety of use cases, data peculiarities, and potential uses has had a substantial impact in how they have been created, extended, and applied. This situation is closely related to the global adoption of these languages and their associated tools. The large number of languages, compliant tools, and usually the lack of information of the combination of both leads users to use other techniques to construct Knowledge Graphs. Often, users choose to create their own ad hoc programming scripts that suit their needs. This choice is normally less reproducible and maintainable, what ultimately affects the quality of the generated RDF data, particularly in long-term scenarios. In this paper, we present the Conceptual Mapping, built as an ontology and designed to represent the expressiveness of existing mapping languages to construct Knowledge Graphs. We devise with this ontology an enhancement to the interoperability of existing mapping languages, which may be achieved in different ways by ensuring the representation of other mappings and by allowing translation among them. This language is built as a result of a thorough analysis of the features and capabilities of current mapping languages, which is presented as a comparative framework.

Keywords: Mapping Languages, Ontology Description, Mapping Translation

1. Introduction

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

In order to countermeasure the heterogeneity of data on the Web, a large number of techniques that translate such data into RDF have been proposed [3, 4], as well as others that assist in tasks related to translation (e.g., facilitating the consumption of heterogeneous data as if they were expressed in RDF [5]). These techniques may follow, namely, two approaches: RDF materialization, that consists of translating data from one or more heterogeneous sources into RDF; or Virtualization, (Ontology Based Data Access) [6] that consists in 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 approaches rely on an essential element, a mapping document, which is the key-enabler for performing the required translations.

Mapping languages represent the relationships between the structure or the model of heterogeneous data and an RDF version following an ontology, i.e., the rules on how to translate from non-RDF data into RDF. This data can be originally expressed in a variety of formats, such as tabular, JSON, or XML. Due to the heterogeneous nature of data, the wide corpus of techniques and specific requirements that some scenarios may impose, an increasing number of mapping lan-

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

guages have been proposed [8–10]. The differences among them are usually based on three aspects: (a) the focus on one or more particular data formats, e.g., the W3C Recommendations R2RML focuses on SQL tabular data [11]; (b) a specific feature they address, e.g. SPARQL-Generate [12] allows the definition of functions in the mapping for cleaning or linking the generated RDF data; or (c) if they are designed for a particular technique or scenario that has special requirements, e.g. the WoT-mappings [13] where designed as an extension of the WoT standard [14] to be used as part of the Thing Descriptions [15].

As a result, the diversity of mapping languages allows the translation of heterogeneous data into RDF in many different scenarios [16–19]. Nevertheless, the existing techniques usually implement just one mapping language, and sometimes not even the whole 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 needed requirements [21]. Some scenarios require a combination of mapping languages because of their differential features, which entails using different techniques. In many cases, this diversity leads to ad hoc solutions that reduce reproducibility, maintainability, and reusability [22].

We propose in this paper the Conceptual Mapping, a novel language that gathers the expressiveness of existing mapping languages. We believe that this language, being designed to represent the other languages, can facilitate interoperability among them, representing them and potentially serving as a common interchange language to allow mapping translation [23]. The proposed language is the result of the analysis of the features of state-of-the-art languages. This analysis studies how languages describe the access to data sources, how triples are created and their distinctive features; and is presented as a comparative framework.

The mapping language presented in this article is implemented as an ontology. Following good methodological practices, it has been developed reusing existing standards such as DCAT [24] and WoT Security[1]. The full mapping language specification is publicly available under the licence CC BY-NC-SA 4.0. Several examples of usage, and comparisons with other languages, are also available in the ontology portal[2].

The rest of this article is structured as follows. Section 2 provides an overview of relevant works revolving around mapping languages and mapping translation. Section 3 presents a thorough analysis and a comparative framework of the different mapping languages and their features. Section 4 describes in detail the ontology that represents with examples the proposed mapping language. Section 5 discusses the challenges and potential uses of the proposed language. Finally, Section 6 presents the conclusions derived from the work carried out in this article.

[1] https://www.w3.org/2019/wot/security
[2] https://w3id.org/conceptual-mapping

Table 1
Analyzed mapping languages and their corresponding references

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

2. Related Work

In this section, first the current scene of mapping languages is described, regardless the type of technique that uses them, i.e., RDF materialization or virtualization. Then, previous works comparing mapping languages are surveyed. Finally, since the mapping language presented in this paper will potentially enable a round-trip translation among mappings, existing related works about language translation are reported.

2.1. Mapping languages

The different scenarios where mapping languages are used and their specific requirements have led to the creation of several mapping languages and tailored do-

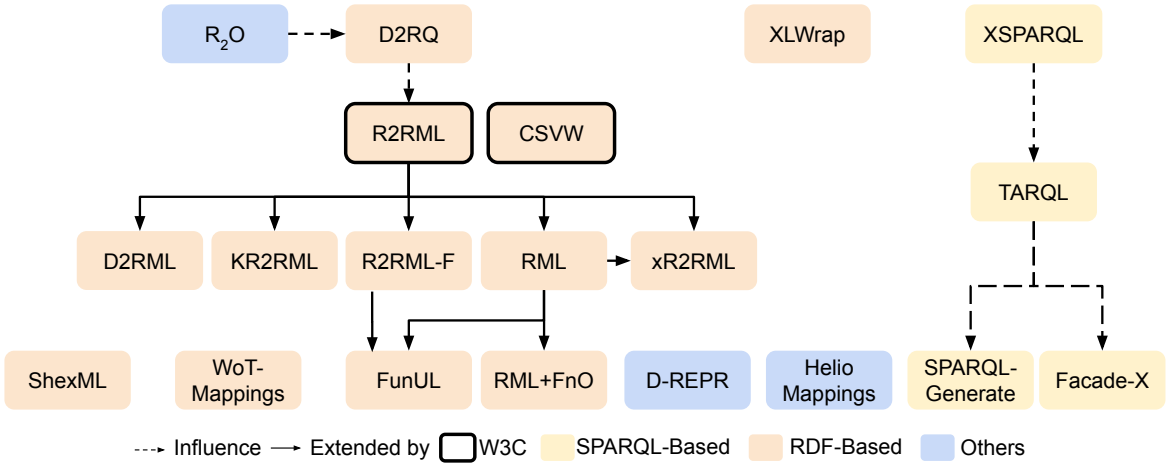


Fig. 1. Existing mapping languages and their relationships

main extensions. This section presents and describes existing mapping languages, summarized in Table 1. Depending on their syntax, they can be classified into: RDF-based, SPARQL-based, and based on other formats. It is worth to mention that some of these mapping languages have become W3C recommendations, namely, R2RML [11] and CSVW [34].

RDF-based. The specification of this type of languages is implemented through the definition of a vocabulary based on RDF, and thus, the mappings are expressed in RDF. 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 data stored in relational databases to RDF. This language is heavily influenced by the previous languages (R_2O [26] and D2RQ [25]). Some serializations (e.g. SML [40], OBDA mappings from Ontop [41]) and several extensions of R2RML were developed in the following years of its release: R2RML-f [30] extends R2RML to include functions to be applied over the data; RML [8] and its serialization YARRRML [42] 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)[3] [17]; FunUL [29] proposes an extension to also incorporate functions, but focusing on the CSV format; KR2RML [28] 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, 27] extends R2RML and RML to include noSQL databases and incorporates more features to handle tree-like data; D2RML [31], 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 [33] is focused on transforming spreadsheets in different formats and CSV files. CSVW [34] enables annotating tabular data on the web with metadata, but also supports the generation of RDF. ShExML [10, 32] uses Shape Expressions (ShEx) [43] to map data sources in RDBs, CSV, JSON, XML and RDF via SPARQL queries. Finally, WoT Mappings [13] are oriented to be used in the context of the Web of Things.

SPARQL-based. The specification of this type of languages is usually based on, or is an extension of, the SPARQL query language [44]. XSPARQL [35] merges SPARQL and XQuery to transform XML to RDF. TARQL [36] uses the SPARQL syntax to generate RDF from CSV files. SPARQL-Generate [12] is able to generate 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" [37]. It does not extend the SPARQL language, instead it overrides the SERVICE operator.

Others. This group gathers other mapping languages that are not based on RDF or SPARQL, instead they

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

rely on a different format. The Helio mapping language [38] is based on JSON and provides the capability of using functions for data transformation and data linking [45]. D-REPR [39] focuses on describing heterogeneous data with JSONPath and allows the use of data transformation functions.

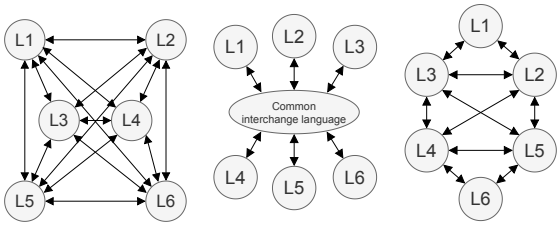
2.2. Language comparison

As the number of mapping languages grew and their adoption became wider, comparisons among these languages were inevitably done. This is the case of, for instance, SPARQL-Generate [12], that is compared with RML in terms of query/mapping complexity; and ShExML [10], that is compared with SPARQL-Generate and YARRRML from a usability perspective.

Some studies dig deeper, providing more complex comparison frameworks. Hert et al. [46] 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 of 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 remarking the need of building a more complete and precise comparative framework and asking for a more active involvement from the community to build it. To the best of our knowledge, there is no single comprehensive work in the literature comparing all existing languages.

2.3. Language translation

There are several approaches that carry out translations among a set of languages. In this work we highlight three of them [47]: Peer-to-peer translation (Fig. 2a), that supports ad-hoc translating solutions between pairs of languages; common interchange language (Fig. 2b), where a language serves as an intermediary among several languages; and families of languages (Fig. 2c), that considers sets of languages and



(a) Peer-to-peer translation approach. (b) Common interchange language. (c) Family of languages approach.

Fig. 2. Types of language translations (Adapted from [48]).

translations between the representatives of each set. In this work, we adopt the second approach, i.e., creating a common shared language that holds their expressiveness to act as a common interchange language.

There have been more situations in the semantic web domain when the need of translating languages has emerged. Before the standarization of the Web Ontology Language (OWL) [49, 50], there were several languages, from those based on first-order logic (e.g., KIF, FLogic) to markup languages (e.g. DAML+OIL, RDF(S)), to describe and create ontologies. Ontolingua [51] was created to write ontologies and interoperate between different first-order logic representations. Additional translation systems were developed later on [47, 48, 52].

Regarding mapping languages, there are currently some implementations that translate unidirectionally pairs of mapping languages. ShExML and YARRRML in their respective online editors[4,5] enable translation to RML. Another case is when tools implement RML/R2RML mapping translation into the language they are designed to parse; such is the case of Helio[6] and SPARQL-Generate[7], that translate from RML; and Ontop, that translates R2RML into its proprietary language, OBDA mappings [41]. Such translation makes it possible to extend the outreach of the tool.

Another case is presented in Mapeathor [53], that takes the mapping rules specified in spreadsheets in a language-independent way and transforms them into a mapping in either one of the following languages: R2RML, RML or its serialization YARRRML. Finally, we remark the case where tools provide a set of opti-

[4] http://shexml.herminiogarcia.com/editor/

[5] https://rml.io/yarrrml/matey/#

[6] https://github.com/oeg-upm/helio/wiki/Streamlined-use-cases#materialising-rdf-from-csv--xml-and-json-files-using-rml

[7] https://github.com/sparql-generate/rml-to-sparql-generate

mizations on the construction of RDF graphs exploiting the translation of mapping rules, such is the case of Morph-CSV [54] and FunMap [55]. Morph-CSV first performs a transformation over the tabular data with RML+FnO mappings and CSVW annotations, and outputs a database and R2RML mappings ready to be transformed by an R2RML-compliant tool. FunMap takes an RML+FnO mapping, performs the transformation functions indicated, outputs the parsed data and generates a function-free RML mapping.

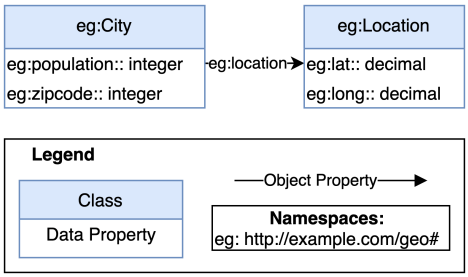
3. Comparison framework

In this section, we present a comparison framework that collects and analyzes the main features included in mapping language specifications. It aims to fill the aforementioned gap that exists in the literature about the comparison of languages. The diversity of the languages that have been analysed is crucial to extract relevant features and requirements. For this reason the framework analyses languages from the three categories identified in Section 2.

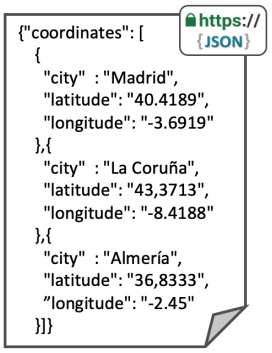
The following RDF-based languages are included: R2RML [11], RML [8], KR2RML [28], xR2RML [9], R2RML-F [30], FunUL [29], XLWrap [33], WoT mappings [13], ShExML [10], CSVW [34], and D2-RML [31]. Analyzed SPARQL-based languages are: XSPARQL [35], TARQL [36], and SPARQL-Generate [12]. Finally, we selected the following languages based on other formats: Helio Mappings [38] and D-REPR [39]. We decided not to include serializations; nor D2RQ [25] and R_2O [26], since these languages were superseded by R2RML, which is included in the comparison. Facade-X [37] was not published and available at the time the presented analysis was carried out, for that reason it is not included.

The framework presented has been built as a result of analysing 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 about data sources, general features for the construction of RDF graphs, and features related to triple creation. 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 of these tables are presented in Appendix A.

Throughout the section, there are examples showing how different languages use the analyzed features.



(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 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[8].

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

```
<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.

3.1. 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 the 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 of the 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 for WoT mappings, which is 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 that each language provides 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` for describing 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 ShExML, RML, SPARQL-Generate, Helio, and D2RML (e.g. `QUERY` in ShExML, SPARQL service description[9] 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).

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

```
<#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: 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.

3.2. 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 analyses 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 form one file can be used) the term is generated; if *Hierarchy Iteration* over different nested levels in tree-like formats is allowed; and whether *Functions* can be used to perform transformations on the data to create the term (e.g. lowercase, toDate, etc).

Subject Generation. Subjects can be either 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}-{name}"`), ShExML, xR2RML, SPARQL-Generate, and Helio with different formats. For instance, 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, Helio and xR2RML. About a third of the languages allow hierarchy iterations (ShExML, XSPARQL, KR2RML, SPARQL-Generate, D-REPR, 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 three languages do not allow dynamic predicates (WoT mappings, ShExML, and XLWrap). For those which 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, 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 able to generate correctly the zipcodes (Listing 5), but it lacks the ability of generating correctly the URI without spaces. RML+FnO is able to do the opposite (Listing 4).

```
mappings:
  Locations:
    sources:
      - [locations_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.

3.3. 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 yo 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*, 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), and *provenance* (i.e. the ontology that the mapping uses to describe the data).

Most of the RDF-based languages allow the static assignment to named graphs; neither SPARQL-based languages nor languages based on other schemes consider this feature. R2RML, RML, R2RML-F, FunUL and D2RML enable also dynamic definitions (e.g., `rr:graphMap` in R2RML and in its abovementioned extensions). Theoretically, the rest of R2RML exten-

sions 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, 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, 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. Finally, no language considers the specification of its provenance.

Linking Rules. Linking rules refers 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 of languages by joining one ore more data fields. Five languages do not allow these links: TARQL, CSVW, KR2RML, WoT, 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 Levenshtein distance and Jaro-Winkler, e.g., Helio.

Transformation functions. Applying functions in mappings allow transforming data before it is semantified. 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 analysed 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[10]). 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, 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 the year 2020. In addition, the instances of the classes `eg:City` and `eg:Location` can be joined using the city name, present in both data sources. However, the names don't exactly match ("Almería" and "Almeria"; "A Coruña" and "La Coruña"), that is why a distance metric is required to match the cities with a threshold of 0.75. The Helio mapping is not able to describe the condition for 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 removing 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.

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

```
{"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. Mapping Language Ontology

This section first describes the methodologies and procedures followed to implement the mapping language as an ontology. Then, the terms defined in the ontology are described. The section ends with an example presenting how the ontology can be populated with several use cases.

4.1. Methodology

The ontology was developed following the guidelines provided by the Linked Open Terms (LOT) methodology. LOT is a lightweight methodology for the development of ontologies and vocabularies [56]. It is based on the previous NeOn methodology [57] and includes four major stages: Requirements Specification, Implementation, Publication, and Maintenance.

At the beginning of the requirements identification stage, the goal and scope of the ontology are defined. Following this, the domain is analyzed in more detail by looking at documentation, data that has been published, standards, formats, etc. In addition, the use cases and user stories are identified. Then, the requirements in the form of competency questions and statements are specified and validated by the stakeholders.

The Conceptual Mapping ontology defines a model that abstracts the features that are present in state-of-the-art mapping languages (as shown in Table 1); while considering their heterogeneity. The context of this model encompasses Ontology Based Data Access (OBDA) and Ontology Based Data Integration (OBDI), where data from several heterogeneous sources need to be accessible according to a unified and global view. Mappings are used to describe the relationships between the global view (i.e., ontologies) and the data source schemes. The diversity of available mapping languages motivates the construction of a language that gathers their features to, in the end, be able to represent them. This can potentially enhance language interoperability and translation, as has been described by Corcho *et al.* [23].

A real user story within this use case is the development of GTFS-Madrid-Bench [20], a benchmark to evaluate knowledge graph construction engines in the transport domain. Different mapping languages are supported by these engines and include R2RML, xR2RML, RML, and CSVW to provide annotations for the CSV datasets. All of these mapping rules represent the same correspondence rules between the ontology and the GTFS data source model. Mappings for the benchmark were created originally in YARRRML, translated to R2RML, and then, depending on the engine evaluated, were in turn translated to languages such as OBDA (Ontop engine's mapping language) and xR2RML. This use case would certainly benefit from mapping translation, since there would be no need of maintaining the same mapping rules in four different languages, just in one.

The LOT methodology defines the goal of the Implementation stage as to build the ontology using a formal language, based on the ontological requirements identified by the domain experts [56]. This stage is developed iteratively through several sprints and it is comprised of the Conceptualization, Encoding, and Evaluation processes. During the conceptualization process, an ontology model is built and represented in a graphical language. The Conceptual Mapping ontology conceptualization is represented graphically with the Chowlk notation [58] (as shown in Fig. 4). This process required the analysis of all of the language features, described in detail in Section 3. The initial iterations included the basic and common fea-

tures of all languages, and later features of specific languages were included in the model. Moreover, other ontologies were reused: Data Catalog (DCAT) and the Web of Things Security (WoT) ontologies; as well as classes and properties from the RDF(S) and OWL specifications. Predefined values for certain properties are represented using the Simple Knowledge Organization (SKOS) vocabulary: the functions and protocols that a mapping may use.

During this process, the ontology was partially validated through examples, that is, by representing with the Conceptual Mapping model sets of mapping rules that were developed in different mapping languages. Once a stable version was reached, the ontology was encoded in OWL. As a final implementation activity, the ontology was evaluated using OOPS! [59]; some minor issues were found regarding reused properties and classes, which were fixed.

The ontology was then published [11] and HTML documentation was generated with Widoco [60], using OnToology [61]. As for maintenance, the ontology is available in a GitHub repository [12].

4.2. Ontology description

The conceptual model of the ontology is shown in Fig. 4. It is divided into two parts which are denoted as Statements and Data Sources.

Data sources. A data source (`DataSource`) describes the data that will be mapped to an ontology using its defined rules. 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 (`Periodic Source`). Details of the data source access are represented with the data access service class (`DataAccess Service`), that 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` 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[13]) 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.

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 (`CartessianProductCombination`).

A source frame corresponds to a data source (with `hasDataSource`) and defines which is the data in the source that is retrieved and how it is fragmented (with `expression`). Among others, JSONPaths, XPaths, queries or regular expressions can be expressed with this feature. 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

[11] https://w3id.org/conceptual-mapping
[12] https://github.com/oeg-upm/Conceptual-Mapping
[13] https://www.w3.org/2019/wot/security

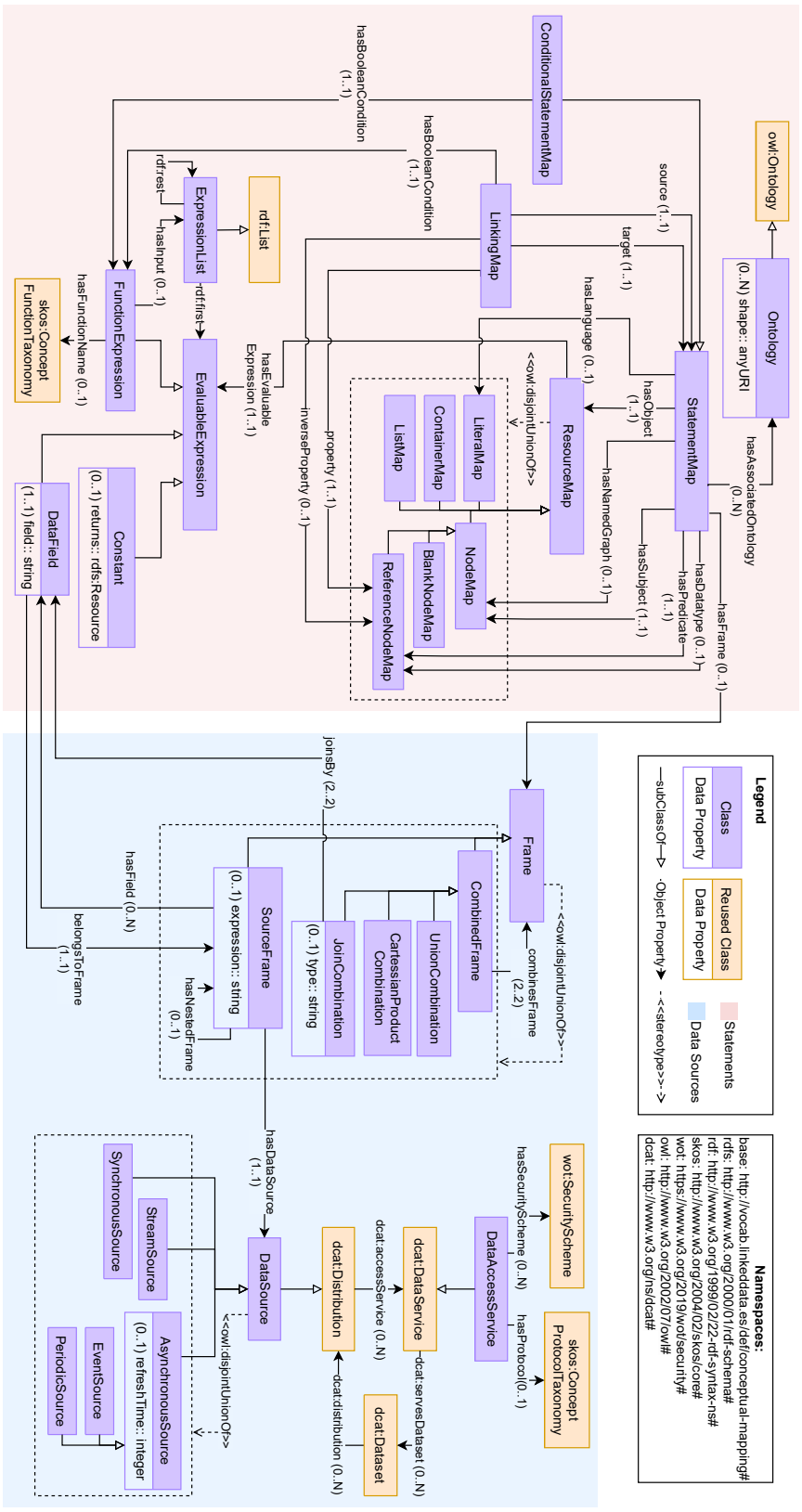


Fig. 4.: Visual representation of the Conceptual Mapping ontology created using the Chowlk diagram notation [58].

(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) which 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. A statement map can also specify the ontology (Ontology) it uses, and can define shapes (shapes) for its restrictions.

Resource maps are expressed with an evaluable expression (EvaluableExpression) that may be a constant value (Constant), a function expression (Function Expression), 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. 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.

4.3. 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 use the data sources "coordinates.json" (Fig. 3b) and "cities"(Fig. 3c) as input and the ontology depicted in Fig. 3a as reference, in order to create the output RDF shown in Listing 1.

```
# Locations
:FrameLoc a cm:SourceFrame;
  cm:expression "$.coordinates[*]";
  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: Data sources.

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

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

:StatementLoc2 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:returns 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:returns 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:returns "http://ex.com/city/"] :
        city_ns)].

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

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

Listing 9: Mapping rules to describe regular statements.

```
:StatementCit2 a cm:ConditionalStatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:returns eg:
        population] ];
  cm:object [ a cm:Literal;
    cm:hasEvaluableExpression  [cm:returns :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.

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 ("text/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" has also 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.

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 generate correctly 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/".

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".

5. Discussion

The wide variety of use cases, data peculiarities, and potential uses has had a substantial impact in how mapping languages have been created, extended, and applied. We show this diversity in the comparison framework (Section 3 and Appendix A), where we describe the distinct features of each language specification, their weaknesses, and strengths, taking into account the purpose and context in which they were developed. This situation is directly related to the adoption of these technologies. The large number of mapping language choices and compliant tools, and the lack of information on the valid combinations of languages and tools, lead users to use other techniques to create Knowledge Graphs. Often, users choose to create their own ad hoc programming scripts to suit their needs. This choice is less reproducible and maintainable, and ultimately affects the quality of the RDF data that has been generated, particularly in long-term scenarios.

In this work, we present a Conceptual Mapping that represents the common and specific features of current languages. The purpose of this language is to enhance the interoperability of existing mapping languages, achieved in different ways: (1) by ensuring the representation of existing mapping languages and (2) by allowing translation among mappings.

The potential of the Conceptual Mapping can be applied to several different use cases. Focusing on language representation, the Conceptual Mapping can also serve to describe current mappings in terms of rule information and metadata. As mentioned in the ontology description (Section 4), the Conceptual Mapping is able to define the provenance of the mapping (i.e., the ontology it follows) and the applicable shapes to validate it. Due to the extensible character of the language, it can also represent additional metadata: who created the mapping, when was it created and modified, etc., such as any ontology metadata. This use can surely benefit mapping governance, a concept that has not been developed so far, but will surely gain importance as the adoption of these technologies increases.

Focusing on language translation, the Conceptual Mapping can ease the Knowledge Graph generation process. Users will not need to learn several languages and how to use their corresponding tools, especially when dealing with use cases that are not addressed by a single specification. They will be allowed to define the mapping rules in the specification they already know, and then perform a translation to use the engine that fits better their own use case.

This can be also applied to evaluation systems that want to provide support for several mapping specifications, such as benchmarks or conformance test cases, which may benefit from the Conceptual Mapping. Before this integration, the definition of each set of test cases (e.g., R2RML [62] or RML [63]) was done mainly manually and without following any standard procedure. Besides the great amount of effort in doing this task, the defined test cases are not comparable in terms of common features, which again, affects understanding the trade-offs of each specification. Having the possibility of translating the test cases between different mapping languages can guarantee a fair comparison while reducing the amount of manual effort.

Another clear use case where the need of the Conceptual Mapping arises is when performing evaluations of different tools that parse different languages, such was the case of GTFS-Madrid-Bench [20]. This benchmark uses mappings in R2RML, RML, CSVW and xR2RML that were created manually. This fact resulted in the investment of a great amount of effort and time to have mappings that represent the same rules, while also having to correct errors and typos coming from manual work. Having tools that allow translation across these languages, ensuring information preservation, would facilitate these tasks and reduce errors.

However, the reality of actual mapping translation is hard. As Section 2 describes, the different language specifications follow different schemes (e.g.,

RDF-based, SPARQL-based, etc.), RDF and SPARQL among them. The translation among RDF-based languages can be performed by using current techniques, such as ontology alignment [64]. Tables of correspondence between classes and properties can be done to clearly specify the relationships between the ontologies followed by these languages. Although differently, this strategy can also be applied to languages with other schemes, such as Helio, which follows a clearly defined scheme for writing rules based on JSON that can be easily matched to entities in RDF-based languages. SPARQL-based languages suppose a greater challenge, due to their intrinsic flexibility and considerable differences with RDF-based languages.

In terms of sustainability, we guarantee that the ontology will evolve incorporating new features of the mapping languages, ensuring transparency and the correct translation among them. For example, at the moment of writing, the RML mapping language is evolving towards an updated specification throughout the W3C Community Group on Knowledge Graph Construction[14]. It will incorporate, among others, the possibility of generating RDF-star datasets [65]. The Conceptual Mapping will, therefore, include such a new feature to keep representing the entire expressiveness of the language.

6. Conclusion and Future Work

In this paper, we present the Conceptual Mapping, an ontology-based conceptual model that aims to gather the expressiveness of current mapping languages. Firstly, we conduct an extensive analysis of the state-of-the-art mapping specifications, enhancing the understanding between the current mapping languages and expanding previous studies on the comparison of language features. The creation of this comparison framework allows us to develop a unique model that not only integrates the common features among the languages, but also other ones that are relevant for integrating heterogeneous data sources across the web. We believe that this contribution potentially serves as a common interchange language for mapping translation purposes [23], being one of the first steps towards a new generation of Knowledge Graph construction techniques, where mapping rules are first-class citizens during the data integration process.

[14] http://w3.org/community/kg-construct/

Our future work lines include the integration of the mapping translation step into the common workflow for constructing virtual and materialized Knowledge Graphs, using the conceptual model as the core resource for carrying out this process. Additionally, we also want to integrate into our previous work about mapping rules management, MappingPedia [66], 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 [67, 68].

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).

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.L. Gayo, R. Navigli, S. Neumaier et al., Knowledge graphs, *ACM Computing Surveys (CSUR)* **54**(4) (2021), 1–37.

[3] F. Michel, J. Montagnat and C.F. Zucker, A survey of RDB to RDF translation approaches and tools, PhD thesis, I3S, 2014.

[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 (2021).

[5] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati and M. Zakharyaschev, Ontology-based data access: A survey, International Joint Conferences on Artificial Intelligence, 2018.
[6] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, *Journal on data semantics X* (2008), 133—-173.
[7] A. Chebotko, S. Lu and F. Fotouhi, Semantics preserving SPARQL-to-SQL translation, *Data & Knowledge Engineering* **68**(10) (2009), 973–1000.
[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: *LDOW*, 2014. ISSN 16130073.
[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)*, 2015, pp. 443–454.
[10] ShExML: improving the usability of heterogeneous data mapping languages for first-time users, *PeerJ Computer Science* **6** (2020), e318. https://peerj.com/articles/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, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10249 LNCS** (2017), 35–50. ISBN 9783319580678.
[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), 822.
[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.
[17] B. De Meester, W. Maroy, A. Dimou, R. Verborgh and E. Mannens, Declarative data transformations for linked data generation: The case of DBpedia, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10250 LNCS** (2017), 33–48. ISBN 9783319584508.
[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** (2018), 16–32.
[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: *International Semantic Web Conference*, Springer, 2020, pp. 294–310.
[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.
[21] B. De Meester, W. Maroy, A. Dimou, R. Verborgh and E. Mannens, Declarative data transformations for linked data generation: The case of DBpedia, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10250 LNCS** (2017), 33–48. ISBN 9783319584508.
[22] A. Iglesias-Molina, D. Chaves-Fraga, F. Priyatna and O. Corcho, Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings., in: *SWAT4HCLS*, 2019.
[23] O. Corcho, F. Priyatna and D. Chaves-Fraga, Towards a new generation of ontology based data access, *Semantic Web* **11**(1) (2020), 153–160.
[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: *Proceedings of the 3rd international semantic web conference (ISWC2004)*, Vol. 2004, Proceedings of ISWC2004, 2004.
[26] 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, 2004.
[27] F. Michel, L. Djimenou, C.F. Zucker and J. Montagnat, xR2RML: Relational and non-relational databases to RDF mapping language, PhD thesis, CNRS, 2017.
[28] J. Slepicka, C. Yin, P.A. Szekely and C.A. Knoblock, KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources., in: *Cold*, 2015.
[29] 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*, 2016, pp. 267–275.
[30] C. Debruyne and D. O'Sullivan, R2RML-F: towards sharing and executing domain logic in R2RML mappings, in: *LDOW@ WWW*, 2016.
[31] A. Chortaras and G. Stamou, D2RML: Integrating Heterogeneous Data and Web Services into Custom RDF Graphs., in: *LDOW@ WWW*, 2018.
[32] 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*, 2021.
[33] A. Langegger and W. Wöß, XLWrap–querying and integrating arbitrary spreadsheets with SPARQL, in: *International Semantic Web Conference*, Springer, 2009, pp. 359–374.
[34] J. Tennison, G. Kellogg and I. Herman, Model for tabular data and metadata on the web, *W3C Recommendation* (2015).
[35] 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.
[36] Tarql: SPARQL for Tables, 2019. http://tarql.github.io/.

[37] E. Daga, L. Asprino, P. Mulholland and A. Gangemi, Facade-X: an opinionated approach to SPARQL anything, *arXiv preprint arXiv:2106.02361* (2021).

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

[39] 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*, ACM, 2019, pp. 189–196. ISBN 9781450370080. https://dl.acm.org/doi/10.1145/3360901.3364449.

[40] C. Stadler, J. Unbehauen, P. Westphal, M. Sherif and J. Lehmann, Simplified RDB2RDF mapping, *CEUR Workshop Proceedings* **1409** (2015).

[41] M. Rodriguez-Muro and M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, *Journal of Web Semantics* **33** (2015), 141–169.

[42] P. Heyvaert, B. De Meester, A. Dimou and R. Verborgh, Declarative Rules for Linked Data Generation at your Fingertips!, in: *Proceedings of the 15th ESWC: Posters and Demos*, 2018.

[43] 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*, 2014, pp. 32–40.

[44] 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).

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

[46] M. Hert, G. Reif and H.C. Gall, A Comparison of RDB-to-RDF Mapping Languages Categories and Subject Descriptors, *The 7th International Conference on Semantic Systems (I-Semantics '11)* (2001), 25–32. ISBN 9781450306218.

[47] J. Euzenat and H. Stuckenschmidt, The 'family of languages' approach to semantic interoperability, *Knowledge transformation for the semantic web* **95** (2003), 49.

[48] O. Corcho and A. Gómez-Pérez, A layered approach to ontology translation with knowledge representation, PhD thesis, UPM, 2004.

[49] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein et al., OWL Web Ontology Language, W3C Recommendation 10 February 2004, *https://www.w3.org/TR/owl-ref/* (2004).

[50] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph et al., OWL 2 Web Ontology Language, W3C Recommendation 11 December 2012, *https://www.w3.org/TR/owl2-primer/* (2012).

[51] T.R. Gruber, A translation approach to portable ontology specifications, *Knowledge acquisition* **5**(2) (1993), 199–220.

[52] D. Dou, D. McDermott and P. Qi, Ontology translation on the semantic web, in: *Journal on data semantics II*, Springer, 2005, pp. 35–57.

[53] A. Iglesias-Molina, L. Pozo-Gilo, D. Doña, E. Ruckhaus, D. Chaves-Fraga and Ó. Corcho, Mapeathor: Simplifying the specification of declarative rules for knowledge graph construction, in: *ISWC (Demos/Industry)*, 2020.

[54] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M.-E. Vidal and O. Corcho, Enhancing virtual ontology based access over tabular data with Morph-CSV, *Semantic Web* (2021), 1–34.

[55] S. Jozashoori, D. Chaves-Fraga, E. Iglesias, M.-E. Vidal and O. Corcho, FunMap: Efficient execution of functional mappings for knowledge graph creation, in: *International Semantic Web Conference*, Springer, 2020, pp. 276–293.

[56] R. García-Castro, A. Fernández-Izquierdo, C. Heinz, P. Kostelnik and F. Serena, D2.2 Detailed specification of the semantic model, Technical Report, Universidad Politécnica de Madrid (UPM), VICINITY Project, 2017. https://vicinity2020.eu.

[57] 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.

[58] S.C. Feria, R. García-Castro and M. Poveda-Villalón, Converting UML-based ontology conceptualizations to OWL with Chowlk, in: *ESWC (P&D)*, 2021.

[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.

[60] D. Garijo, WIDOCO: a wizard for documenting ontologies, in: *International Semantic Web Conference*, Springer, 2017, pp. 94–102.

[61] 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.

[62] B. Villazón-Terrazas and M. Hausenblas, R2RML and Direct Mapping Test Cases, W3C Note, W3C, 2012, http://www.w3.org/TR/rdb2rdf-test-cases/.

[63] P. Heyvaert, D. Chaves-Fraga, F. Priyatna, O. Corcho, E. Mannens, R. Verborgh and A. Dimou, Conformance test cases for the RDF mapping language (RML), in: *Iberoamerican Knowledge Graphs and Semantic Web Conference*, Springer, 2019, pp. 162–173.

[64] M. Ehrig, *Ontology alignment: bridging the semantic gap*, Vol. 4, Springer Science & Business Media, 2006.

[65] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga and A. Dimou, RML-star: A Declarative Mapping Language for RDF-star Generation, in: *International Semantic Web Conference - Demos&Industry*, 2021.

[66] F. Priyatna, E. Ruckhaus, N. Mihindukulasooriya, Ó. Corcho and N. Saturno, Mappingpedia: A collaborative environment for R2RML mappings, in: *European Semantic Web Conference*, Springer, 2017, pp. 114–119.

[67] 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*, 2010, pp. 227–238.

[68] 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.

Appendix A. Framework Comparison of Existing Mapping Languages

Table 2
Data retrieval and data source expression for the analysed mapping languages. (*) 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
Retrieval of data	Streams	false	false	false	false	false	true*	false	false	true	false	false	false	false	false	false	false
	Synchronous sources	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 (RDB)*	Basic (RDB)	Basic (RDB)	Basic (RDB)	-	Basic (RDB)	-	API Key, OAuth2, Bearer, Basic	API Key, OAuth2, Bearer, Basic	-	-	Basic (RDB)
	Encoding	false	false	true*	true	false	true	false	false	false	false	false	true	true	false	false	false
	MIME Type	false	false	false	false	false	false	false	false	false	false	false	true	true	false	false	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
	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
	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

Table 3: Features for subject, predicate, and object generation of the studied mapping languages.

Feature & Language			ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FunUL	Helio	WoT	D-REPR	XLWrap	D2RML
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
	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
		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
		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	true	true	false	false	false	true	true	false	true	false	false	false	false	true	false	true
		Functions	-	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
	Dynamic	RDF Resource	-	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
		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
		Hierarchy Iteration	false	false	false	false	false	true	true	false	false	false	false	false	false	true	false	true
		Functions	-	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
	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
		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
		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	true	true	false	false	false	true	true	false	true	false	false	false	false	true	false	true
		Functions	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

Table 4: Statements, linking rules, and function properties of the studied mapping languages.

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