R2RML and RML Comparison for RDF Generation, their Rules Validation and Inconsistency Resolution

Anastasia Dimou[0000−0003−2138−7972]

Ghent University – imec – IDLab, Belgium

Abstract. In this paper, an overview of the state of the art on knowledge graph generation is provided, with focus on the two prevalent mapping languages: the W3C recommended R2RML and its generalisation RML. We look into details on their differences and explain how knowledge graphs, in the form of RDF graphs, can be generated with each one of the two mapping languages. Then we assess if the vocabulary terms were properly applied to the data and no violations occurred on their use, either using R2RML or RML to generate the desired knowledge graph.

Keywords: Mapping languages · R2RML · RML · Validation

1 Introduction

Knowledge graphs are often generated using rules that apply vocabulary terms to certain data from different data sources. This occurs because most data is still not available in the form of knowledge graphs. Data may have *different structures* (e.g., tabular or hierarchical), appear in *heterogeneous formats* (e.g., CSV, XML or JSON) and are accessed via *heterogeneous interfaces* (e.g., database interfaces or Web APIs) [18]. Therefore, different approaches were proposed to generate knowledge graphs from existing (semi-)structured data.

These different approaches follow different directions, from *custom implementations* [5,21] to *format-specific* [28,27] or *direct mappings* [3,36], but approaches that *detach the rules definition from their execution* prevailed. Detaching the rules renders the rules interoperable between implementations, whilst the systems that process those rules are use-case independent. This improves the knowledge graph generation which becomes more interoperable, reusable, and maintainable. *Mapping languages* were proposed to define such rules to generate knowledge graphs which range from *dedicated languages*, such as R2RML, RML or xR2RML [17,32] to *repurposed languages*, such as SPARQL-Generate [29].

Even though many dedicated mapping languages were proposed, no thorough comparison between those languages was performed so far. Only De Meester et al. [13] presented an initial set of comparative characteristics based on requirements posed by reference works. These characteristics are both *functional* (easy to use [22,29], follow Semantic Web standards [29], fully cover the generation process [25]) and *non-functional* (be extensible [17,25,29] and support general

mapping functionalities [25], nested hierarchies [32] and lists [32]). De Meester et al. [13] observe that multitude of mapping languages allows to support more use cases and conclude that effort should be consolidated on missing features, instead of re-developing existing functionalities.

In this paper, we compare the two most prevalent mapping languages with respect to their syntax: the W3C recommended R2RML [9] and its generalisation RML [17]. We compare what can be done and how in each mapping language. We aim to support data holders to answer the following research question:

Which mapping language between R2RML and RML should one use?

The answer to this research question is explored in section 3.

However, while knowledge graphs are generated by consistently applying terms of certain vocabularies, the rules that define how the vocabularies terms are applied should respect the restrictions imposed by the vocabularies definitions. However, consistently annotating the existing data sources with vocabulary terms is not always straightforward. Inconsistencies may be introduced when the rules are defined [16] and resolutions are required [23], applied to either the rules or the vocabulary terms definitions [23]. We aim to support data holders to answer the following research question:

How can one validate that the vocabulary terms were properly applied to the data using R2RML or RML?

The answer to this research question is explored in section 4.

The remaining of this paper is structured as follows: In section 2, we outline the state of the art on knowledge graph generation using mapping languages, and on knowledge graph validation and inconsistency resolution. Then we focus on R2RML and RML mapping languages and we compare them in section 3. Last, we explain how rules expressed with these mapping languages can be validated and how inconsistencies can be resolved. The paper concludes with a discussion on remaining challenges, open issues and future directions.

2 State of the Art

This paper focuses on (i) *defining* rules to annotate data with vocabulary terms using mapping languages on the one hand, and (ii) *validating* rules that annotate data on the other hand. In section 2.1, the state of the art related to mapping languages is covered, while in section 2.2, the state of the art on validation and resolution approaches for mapping languages is covered.

2.1 Mapping Languages

Different approaches were proposed so far for generating knowledge graphs from (semi-)structured data sources. These approaches range from *custom implementations* [5,31], which appeared mostly in the first years but remain prevalent till

nowadays [6,21], to more *generic approaches*. Such generic approaches originally focused on data with *specific formats*, namely dedicated approaches for, e.g., relational databases [7], or XML [27]. Among the format-specific approaches, solutions for knowledge graph generation from relational databases were the most mature ones and lead to two main directions which also became W3C recommendations: (i) *direct mapping* [3] and (ii) *detached rules* (R2RML) [9].

In the former case, the *direct mapping* defines a simple transformation, providing a basis for defining more complex transformations afterwards. However, *direct mapping* requires defining rules later, for instance using SPARQL queries [2], e.g., Datalift [36], to replace the original predefined annotations with custom ones. In the latter case, the rules definition is detached from their execution. Mapping languages were proposed to define such detached rules, such as D2RQ [7] that lead to the W3C recommended R2RML [9].

However, format-specific approaches, like the ones for relational databases, require data holders to learn and maintain different tools for each data format [17]. Therefore, different solutions were proposed for heterogeneous data sources. The prevalent directions for knowledge graph generation from relational databases, were followed. However, the Direct Mapping cannot be applied for heterogeneous data sources, because each format requires its own Direct Mapping implementation, as it occurs in the case of Datalift [36].

Therefore, most solutions for knowledge graph generation from heterogeneous data sources were focused on detached rules. On the one hand, *dedicated mapping languages* were proposed, e.g., RML [17] that generalises R2RML or xR2RML [32,33] that extends both R2RML and RML. On the other hand, *repurposed mapping languages* were proposed that extend existing languages for other tasks, e.g., SPARQL-Generate [29] that repurposes SPARQL [2]. Nowadays, the most prevalent dedicated mapping languages are defined as extensions of R2RML. RML was the first language to generalise R2RML, but there are more alternative approaches and extensions beyond the originally proposed language.

2.2 Validation and inconsistency resolution

There exists a number of methods to validate knowledge graphs, and detect and resolve inconsistencies. The validation might be applied on either the knowledge graph (*graph-driven validation methods*) or the rules (*rule-driven validation methods*) that define how the knowledge graph is generated.

On the one hand, there are validation methods applied directly to the knowledge graph. Different approaches were proposed to tackle various aspects of Linked Data quality. These approaches can be classified as [12]: (i) *manual* (e.g., [1,4]), (ii) *hard-coded* [24] (iii) *logic-based* (*integrity constraints*, e.g., [34]), (iv) *query-based*, e.g., [20,26] and (v) *rule-based*, e.g.,[12]. These methods rely on the complete knowledge graph and can potentially identify every inconsistency. However, validating the complete knowledge graph is not always possible, especially in time-constrained situations [16]. On the other hand, there are validation methods applied to the rules that generate knowledge graphs [16]. Such methods result in faster execution times, but not all inconsistencies can be identified, as

some of them depend on the actual data values in the graph, e.g., (qualified) cardinality, (inverse) functionality, (a)symmetricity and irreflexivity.

Methods were proposed to detect and resolve inconsistencies. Possible root causes for these inconsistencies include [23]: (i) *raw data* that contain inconsistencies [30]; (ii) *rules* that introduce new inconsistencies by, for example, not using the suitable ontology terms [16,35]; and (iii) *vocabulary definitions* that do not model the domain as desired [35].

3 Mapping Languages: R2RML and RML

In this chapter, we focus on knowledge graph generation with *dedicated mapping languages*. The most noted mapping language is the W3C recommendation R2RML [9], and its most popular generalisation RML [17] for heterogeneous data sources, while xR2RML [33] is their common derivative. RML may be considered broadly adopted because it is the R2RML extension that has the most alternative implementations according to its implementation report[1]: RMLMapper[2], CARML[3], RMLStreamer[4], RocketML[5], SDM-RDFizer[6].

R2RML The Relational to RDF Mapping Language (R2RML) [9] is the W3C recommendation to express customized mapping rules from data in relational databases to generate knowledge graphs represented using the Resource Description Framework (RDF) [8]. R2RML considers a target semantic schema which is a combination of one or more vocabularies. The R2RML vocabulary namespace is `http://www.w3.org/ns/r2rml#` and the preferred prefix is `r2rml`.

RML The RDF Mapping Language (RML) [17] also expresses customized mapping rules, but from data in heterogeneous structures, formats and access interfaces. RML is a superset of R2RML, but always remains backwards compatible with R2RML. RML keeps the mapping rules as in R2RML but excludes its database-specific references from the core model. This way, the mapping rules can be defined in a combined and uniform way, while the input data becomes a set of (one or more) input data sources. The RML vocabulary namespace is `http://semweb.mmlab.be/ns/rml#` and the preferred prefix is `rml`.

In both R2RML and RML, an RDF dataset is generated based on one or more *Triples Maps* (`rr:TriplesMap`). A *Triples Map* defines how a set of RDF triples referring to the same entity, i.e. same subject, are generated. Each *Triples Maps* defines which data is considered (section 3.1), what the *subject* of the RDF triples is (*Subject Map*), and which predicates (*Predicate Map*) and objects (*Object Map*) will be generated to form the different RDF triples (section 3.3).

[1] RML Implementation Report, `https://rml.io/implementation-report`
[2] RMLMapper, `https://github.com/RMLio/rmlmapper-java`
[3] CARML, `https://github.com/carml/carml`
[4] RMLStreamer, `https://github.com/RMLio/RMLStreamer`
[5] RocketML, `https://github.com/semantifyit/RocketRML`
[6] SDM-RDFizer, `https://github.com/SDM-TIB/SDM-RDFizer`

Table 1. R2RML and RML comparison

language	R2RML	RML
prefix	`rr`	`rml`
URI	`http://www.w3.org/ns/r2rml#`	`http://semweb.mmlab.be/ns/rml#`
relational DBs	multiples tables one DB	multiple tables multiple DBs
access interfaces	only ODBC	multiple
other data structures	–	tabular (e.g., CSV, TSV, XLS) hierarchical (e.g., XML, JSON) pair-valued (e.g., wikitext) graphs (e.g., RDF), etc.
integration	materialisation virtualisation	materialisation virtualisation
data transformation	pre-processing	pre-processing inline processing

The following subsections explain the differences between R2RML and RML with respect to (i) the data sources they support (section 3.1), (ii) the RDF terms generation (section 3.2), and (iii) the RDF triples generation (section 3.3).

3.1 Data Source

R2RML and RML support different data sources to generate the RDF datasets.

R2RML A *Triples Map* refers to a *logical table* retrieved from a certain database. A logical table can be a base table, a view, or a valid SQL query, called an *R2RML view* because it emulates a SQL view without modifying the database.

RML Data can originally (i) reside on **diverse locations**, e.g., in files or in a database at the local network, or can be published on the Web; (ii) be accessed using **different interfaces**, e.g. raw files, database connectivity for databases, different interfaces from the Web, such as Web APIs; and (iii) have **heterogeneous structures and formats**, e.g. tabular, such as databases or CSV files, hierarchical, such as XML or JSON format, semi-structured, such as HTML [18].

Table 2. Results of female pole vault for 2019 world championship

rank	name	surname	nationality	mark	notes
1	Anzhelika	Sidorova	Russia	4.95	WL,PB
2	Sandi	Morris	United States (USA)	4.90	SB
3	Katerina	Stefanidi	Greece	4.85	SB
4	Holly	Bradshaw	Great Britain	4.80	

The main difference between R2RML and RML is the type of data they support. R2RML supports homogeneous data, i.e. relational databases, whereas RML supports heterogeneous data. In more details:

R2RML Each *Triples Map* (listing 1.1, line 1) refers to exactly one *Logical Table* (`rr:LogicalTable`), specified by its *table name* (`rr:tableName`). A *Logical Table* is either a SQL base table or view, or an R2RML view. The SQL query result is used to generate the RDF triples. All mapping rules refer to only one database.

```
1  <#PoleVaulters>        rr:logicalTable <#PoleVaultersDBtable> ;
2  <#PoleVaultersDBtable> rr:tableName "poleVaulters" .
```

Listing 1.1. A Triples Map refers to a certain Logical Table specified by its name

RML A *Logical Source* (`rml:LogicalSource`) extends R2RMLs *Logical Table* and defines the data source to be used for the knowledge graph. A broader reference to any input source is considered in RML. Thus, exactly one *Logical Source* is specified (`rml:source`) for each *Triples Map* to indicate the input. For instance, instead of having a *Logical Table* `<#PoleVaultersDBtable>` (listing 1.1, line 1), we have a *Logical Source* `<#CountriesXML>` (listing 1.2, line 1):

```
1  <#Countries>    rml:logicalSource <#CountriesXML> .
2  <#CountriesXML> rml:source <\protect\vrule width0pt\protect\href{http://rml.io/data/semWebSer/countries.xml}{http://rml.io/da
```

Listing 1.2. A Triples Map refers to a Logical Source whose data is in XML format

Such a *Logical Source* may refer to a file that contains the countries with the pole vaulters nationality such as the one that follows (listing 1.3).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <countries>
3    <country country_language="en-uk">Great Britain</country>
4    <country country_language="el">Greece</country>
5    <country country_language="ru">Russia</country>
6    <country country_language="en-us">United States</country>
7  </countries>
```

Listing 1.3. An XML file with countries details

Data Iteration Mapping languages are suitable for knowledge graphs generation because the same rules are applied to multiple data chunks that follow the same structure pattern. While R2RML is focused on tabular structure whose iteration pattern is predefined, the iteration pattern in other data structures is not be predefined, causing yet another difference between R2RML and RML.

R2RML An implied per-row iteration is predefined.

RML The iteration pattern can not always be implicitly assumed, but it needs to be explicitly defined. The *iterator* (`rml:iterator`) (listing 1.4, line 1) determines the iteration pattern over the data source and specifies the data to be considered during each iteration. The *iterator* is not required to be explicitly mentioned in the case of tabular data sources, as the default per-row iteration is implied.

```
1  <#CountriesXML>   rml:iterator "/countries/country" .
```

Listing 1.4. A Logical Source specifies its iterator

3.2 RDF term generation

To generate the RDF triples of an RDF dataset, the RDF terms that define the subject, predicate, object and named graph need to be generated.

Term Maps (`rr:TermMap`) define how RDF terms (IRI, blank node, literal) are generated. They are *constant-*, *template-*, or *column-* (R2RML) or *reference-* (RML) *valued*. The subject, predicate and object of RDF triples are RDF terms.

A *constant-valued Term Map* (`rr:constant`, listing 1.5, line 1) always generates the same RDF term which is by default an IRI.

A *template-valued* Term Map (`rr:template`, listing 1.5, line 2) is a valid string template that contains references and generates an IRI by default.

```
1  <#RDFtermConstant>      rr:constant          ex:score .
2  <#RDFtermTemplate>      rr:template         "http:://ex.com/person/{name}_{surname}" .
```

Listing 1.5. RDF terms are generated with different types of Term Maps

Reference to data fragments R2RML and RML refer differently to the data fragments of the data sources based on which the RDF terms are generated.

R2RML A *column-valued term map* (`rr:column`, listing 1.6, line 1) generates a literal by default that is a column in a given *Logical Table*s row.

```
1  <#RDFtermColumn>      rr:column "name" .
```

Listing 1.6. RDF terms are generated with different types of Term Maps

RML As RML covers heterogeneous data, different references to the data sources apply. A *logical reference* (`rml:reference`) is a valid reference to the data source according to the specified reference formulation. The *Reference Formulation* (`rml:referenceFormulation`) (listing 1.7, line 1), indicates the formulation (for instance, a standard or a query language) used to refer to its data. A *reference-valued term map* generates a literal by default.

```
1  <#CountriesXML>     rml:referenceFormulation ql:XPath .
2  <#RDFtermReference>  rml:reference "name" .
```

Listing 1.7. A Logical Source specifies its Reference Formulation

Language The language of an RDF term which is literal can be defined. R2RML and RML define differently the language, RML extends the R2RML options.

R2RML The language (`rr:language`) of a literal may be optionally defined. R2RML allows only constant values for the language.

```
1  <#CountryName>  rr:column "country" ;
2                  rr:language "en-us" .
```

Listing 1.8. rr:language to define the literal's language

RML RML has a dedicated *Term Map* defining the language, the *Language Map* (`rml:LanguageMap`, listing 1.9, line 2), which extends R2RML's language tag (`rr:language`). The *Language Map* allows not only constant values for defining the language but also references to the data. `rr:language` is considered then an abbreviation for the `rml:languageMap`, if the value is constant.

```
1  <#CountryName>  rml:reference "country_name" ;
2                  rml:languageMap [ rml:reference "@country_language"] ] .
```

Listing 1.9. rml:languageMap to define the literal's language

datatype The datatype (`rr:datatype`) of an RDF term which is literal can be defined as well. Both R2RML and RML define the *datatype* in the same way.

```
1  <#Mark>         rr:column "mark" ;
2                  rr:datatype xsd:positiveInteger .
```

Listing 1.10. rr:datatype to define the literal's datatype

Term type If the default termtype is desired to be changed, the term type (`rr:termType`) is explicitly defined (`rr:IRI`, `rr:Literal`, `rr:BlankNode`).

The *term type* (`rr:termType`) needs to be explicitly mentioned (listing 1.11, line 2) to change the default term type, i.e. `IRI` for *template* and *constant*-valued term maps and `Literal` for *column-* or *reference*-valued term maps. Table 3 summarizes all valid combinations of *Term Maps* and *Term Types* and the type of *RDF term* they generate in each case for both R2RML and RML.

```
1  <#Website>      rml:reference "website" ;
2                  rr:termType rr:IRI .
```

Listing 1.11. rr:datatype to define the literal's datatype

IRIs and constant values are generated in the same way in RML and R2RML. However, the blank nodes are generated differently in R2RML and RML:

R2RML A *Term Map* consists of one *column* reference, *template* or *constant*.

RML A *Term Map* consists of zero or one *reference*, *template* or *constant*. This way, blank nodes can be generated relying on randomly generated IRIs.

```
1  <#RDFtermBlankNode>      rr:termType rr:BlankNode .
```

Listing 1.12. Blank Nodes generated randomly without reference to the data source

In table 3, there is a list of all possible combinations of *Term Maps*, their values and types, as well as what type of RDF term is generated in each case.

Data transformation The data often is not used as they appear in the original data source, but are transformed to generate the desired RDF term. R2RML and RML follow different approaches for transforing the data values.

Table 3. All combinations of Term Maps and Term Types and the RDF terms they generate. The default term type for each term map is in parenthesis (optional to be specified). The most frequent used value and RDF type for each Term Map is in bold. [R2]RML is indicated when the statement is valid for both R2RML and RML.

Term Map	language	value	Term Type	RDF Type
Subject Map `rr:SubjectMap`	[R2]RML	**`rr:template`**	(`rr:IRI`) `rr:BlankNode`	**IRI** BlankNode
		`rr:constant`	(`rr:IRI`)	IRI
		`rr:column`	`rr:IRI`	IRI
	RML	`rml:reference`	`rr:IRI` `rr:BlankNode`	IRI BlankNode
		–	`rr:BlankNode`	BlankNode
Predicate Map `rr:PredicateMap`	[R2]RML	`rr:template`	(`rr:IRI`)	IRI
		`rr:constant`	(`rr:IRI`)	**IRI**
		`rr:column`	`rr:IRI`	IRI
	RML	`rml:reference`	`rr:IRI`	IRI
Object Map `rr:ObjectMap`	[R2]RML	`rr:template`	**IRI** `rr:Literal`	IRI Literal
		`rr:constant`	(`rr:IRI`) `rr:Literal`	IRI IRI
		`rr:column`	(`rr:Literal`) `rr:IRI`	**Literal** IRI

	RML	**`rml:reference`**	`rr:IRI`
			`rr:BlankNode`
			`rr:Literal`
				IRI
				BlankNode
				Literal

Referencing Object Map `rr:RefObjectMap`	[R2]RML	**`rr:parentTriplesMap`**	**IRI** BlankNode	IRI BlankNode
Language Map `rml:LanguageMap`	RML	`rr:template`	`rr:Literal`	n/a
		`rr:constant`	`rr:Literal`	n/a
		`rml:reference`	(`rr:Literal`)	n/a

R2RML A pre-processing approach is considered. If data is desired to be transformed, the transformation occurs with an SQL query or a view.

RML While pre-processing is possible, not all data formats have as powerful query languages as SQL is for relational databases. If pre-processing occurs, the data transformation is not declaratively defined. Therefore, inline processing approaches were proposed, such as FnO [10,14] and FunUL [25].

3.3 RDF triples generation

In this subsection, we explain how RDF terms are combined to define RDF triples. A subject, predicate, object and optionally a named graph are Term Maps that are combined to generate an RDF triple or quad respectively. R2RML and RML follow the same way for generating RDF triples.

Subject The *Subject Map* (`rr:SubjectMap`) defines how RDF terms are generated that are unique identifiers (IRIs [19]) or blank nodes. These RDF terms constitute the subject of all RDF triples generated from the *Triples Map*.

```
1  <#PoleVaulters>     rr:subjectMap     <#Person_SM> .
2  <#Person_SM>.       rr:template       "http::://ex.com/person/{name}"
```

Listing 1.13. The Subject Map of a Triples Map

Predicate-Object A *Predicate-Object Map* (`rr:PredicateObjectMap`) defines the pairs of predicates and objects that characterise the *subject*. It consists of one or more mapping rules to define how the *predicate* (`rr:PredicateMap`) is generated, and one or more mapping rules to define how the *object* (`rr:ObjectMap`) or *Referencing Object Maps* (`rr:ReferencingObjectMap`) is generated.

```
1  <#PoleVaulters>  rr:predicateObjectMap <#Mark_POM> ;
2                   rr:predicateObjectMap <#Nationality_POM>.
```

Listing 1.14. A Triples Map consists of zero or more Predicate Object Maps

Predicate A *Predicate Map* (`rr:PredicateMap`, listing 1.15, line 2) is a *Term Map* (`rr:TermMap`) that defines how the triples predicate is generated.

```
1  # Predicate Object Map with Object Map
2  <#Mark_POM>       rr:predicate          ex:score ;
3                    rr:objectMap          [ rr:column "Mark"] .
4
5  # Predicate Object Map with Referencing Object Map
6  <#Nationality_POM>  rr:predicateMap       <#Country_PM> ;
7                      rr:objectMap          <#Country_ROM> .
```

Listing 1.15. A Predicate Object Map consists of one or more Predicate Maps and one or more Object Maps or Referencing Object Maps

Object An *Object Map* (`rr:ObjectMap`, listing 1.15, line 3) or a *Referencing Object Map* (`rr:ReferencingObjectMap`, listing 1.15, line 7) defines how the triples object is generated. An *Object Map* is a *Term Map* that defines how a resource (IRI or blank node) or a literal will be generated.

Referencing Object A *Referencing Object Map* defines how the object is generated based on the *Subject Map* of another *Triples Map*. If the *Triples Maps* refer to different *Logical Tables*, a join between the *Logical Tables* is required. The *join condition* (`rr:joinCondition`) performs the join exactly as a join is executed in SQL. The join condition consists of a reference to a column name that exists in the *Logical Table* of the *Triples Map* that contains the *Referencing Object Map* (`rr:child`) and a reference to a column name that exists in the *Logical Table* of the *Referencing Object Maps Parent Triples Map* (`rr:parent`).

```
1  # Referencing Object Map
2  <#Country_ROM>   rr:parentTriplesMap <#Country_TM> ;
3                   rr:join  [
4                               rr:cild "nationality" ;
5                               rr:parent "country"] .
```

Listing 1.16. A Referencing Object Map generates an object based on the Subject Map of another Triples Map

In both R2RML and RML, the entity's RDF type and the RDF triple's named graph may be defined on *Subject Map* or *Predicate Object Map* level.

RDF type In both R2RML and RML, the RDF type of an element can be specified with two ways: (i) specifying its class (`rr:class`) in the *Subject Map* (listing 1.17, line 2), or (ii) defining a *Predicate Object Map* (listing 1.18) whose predicate is `rdf:type` (line 1) and its object the desired class (line 2).

```
1  <#Person_SM>  rr:template  "http::://ex.com/person/{name}" ;
2                rr:class     foaf:Person .
```

Listing 1.17. The RDF type of an entity defined in the Subject Map

```
1  <#Mark_POM>   rr:predicate. rdf:type ;
2                rr:object     foaf:Person .
```

Listing 1.18. The RDF type of an entity defined in the Predicate Object Map

Named Graph Each RDF triple is placed into one or more graphs, the unnamed default graph or an IRI-named named graph. A *Subject Map* or *Predicate-Object Map* may have one or more associated *Graph Maps*.

```
1  <#Person_SM>. rr:template. "http::://ex.com/person/{name}_{surname}" ;
2                rr:graph  ex:PersonGraph .
3
4  <#Mark_POM>  rr:predicate          ex:score ;
5               rr:objectMap          [ rr:column "Mark"] ;
6               rr:graphMap           [ rr:constant <\protect\vrule width0pt\protect\href{http://example.com/graph/students}{htt
```

Listing 1.19. The named graph of the RDF triple may be defined in either Subject Map or Predicate Object Map

4 Rules Validation and Inconsistencies Resolution

Either R2RML or RML is considered to define the rules for generating the desired knowledge graph, the rules that define how the knowledge graph is generated, should be validated before they are used to generate the knowledge graph. This way, we prevent the same violations to appear repeatedly, and inconsistencies are resolved in due time. In section 4.1, we explain how the rules can be validated and, in section 4.2 how inconsistencies can be resolved.

4.1 Validation

Rules in both R2RML and RML constitute a knowledge graph, because they have a native RDF representation. Thus, the same set of schema validation patterns, normally applied on the generated knowledge graphs, is also applicable on the knowledge graph of the rules that state how the knowledge graph is generated. Therefore, instead of validating the generated RDF triples, we validate the *Triples* and *Term Maps* that define how the RDF triples should be generated.

RDF triples validation In the case of RDF triples validation, the RDF triples are considered. The RDF triples predicate (`foaf:familyName`) is validated against its subject's class for domain violations (`foaf:Person` in listing 1.20 and `foaf:Document` in listing 1.21) and object (`"Morris"@en-us` in listing 1.20 and `"Morris"^^xsd:integer` in listing 1.21) for range violations.

```
1  ex:Anzhelika_Sidorova a foaf:Person ; foaf:familyName "Sidorova"@en-us .
2  ex:Sandy_Morris       a foaf:Person ; foaf:familyName "Morris"@en-us .
3  ex:Katerina_Stefanidi a foaf:Person ; foaf:familyName "Stefanidi"@en-us .
4  ex:Holly_Bradshaw     a foaf:Person ; foaf:familyName "Bradshaw"@en-us .
```

Listing 1.20. RDF triples without violations.

```
1  ex:Anzhelika_Sidorova a foaf:Document ; foaf:familyName "Sidorova"^^xsd:integer .
2  ex:Sandy_Morris       a foaf:Document ; foaf:familyName "Morris"^^xsd:integer .
3  ex:Katerina_Stefanidi a foaf:Document ; foaf:familyName "Stefanidi"^^xsd:integer .
4  ex:Holly_Bradshaw     a foaf:Document ; foaf:familyName "Bradshaw"^^xsd:integer .
```

Listing 1.21. RDF triples with violations.

While the RDF triples in listing 1.20 do not have any violations, the RDF triples in listing 1.21 do have violations. The RDF triples in the latter case may be corrected, but the root of the violation is not known, unless its provenance is traced. But even then, very fine-grained provenance information is required even on RDF Terms level [15,11] to trace rules that generate violating RDF triples.

rules validation In case of rules validation, the predicate (`foaf:familyName`) is extracted from the *Predicate Map* (listing 1.22, line 7 in the former case and listing 1.23, line 7 in the latter case) and is compared to the ones derived from the corresponding *Subject Map* (listing 1.22, line 5 in the former case and listing 1.23, line 5 in the latter case) and *Object Map* (listing 1.22, line 8 in the former case and listing 1.23, line 8 in the latter case). are identified The properties and classes namespaces are used to retrieve the schemas and generate the test cases.

```
1  <#PoleVaulters>     rr:subjectMap         <#Person_SM> ;
2                      rr:predicateObjectMap <#Name_POM> .
3
4  <#Person_SM>.       rr:template       "http::://ex.com/person/{name}_{surname}" ;
5                      rr:class          foaf:Person .
6
7  <#Name_POM>         rr:predicateMap [ foaf:familyName ];
8                      rr:objectMap    [ rml:reference "surname" ; rr:language "en-us" ] .
```

Listing 1.22. Correct defined rules generate RDF triples without violations.

```
<#PoleVaulters> rr:subjectMap         <#Person_SM> ;
                rr:predicateObjectMap <#Name_POM> .

<#Person_SM>.   rr:template       "http:://ex.com/person/{name}_{surname}" ;
                rr:class          foaf:Document .

<#Name_POM>     rr:predicateMap [ foaf:familyName ];
                rr:objectMap    [ rml:reference "surname" ; rr:dataType xsd:integer ] .
```

Listing 1.23. Incorrect defined rules generate violating RDF triples.

While the rules in listing 1.22 do not seem to generate RDF triples with violations, the rules in listing 1.23 will certainly generate RDF triples with violations. The rules in the latter case may be corrected, and the violations will never appear again, unless new are introduced after the correction.

The RDF triples validation requires performing the validation as many times as the number of the generated RDF triples. In our example, four RDF triples are generated (listing 1.20 and listing 1.21), and zero and eight violations are identified respectively. If there were four million RDF triples, each triple should have been validated, and zero and eight million violations would have been identified respectively. If the RDF triples are corrected, the next time the RDF triples are generated, the same violations will appear.

To the contrary, the rules validation is independent of the number of the generated RDF triples. In our example, a *Predicate Map* is compared with its associated *Subject Map* and *Predicate Map* and two violations are identified. If the violating rules are fixed, every next time that the RDF triples are generated, the RDF triples will not have violations.

Even though assessing the rules can cover many violations related to vocabularies and ontologies used to annotate the data, some schema-related violations depend on how the rules are instantiated on the original data. Therefore, a uniform way of incrementally assessing the quality of a knowledge graph and rules should cover both the rules and the knowledge graph.

4.2 Resolution

Resglass [23] includes a ranking to determine the order with which rules and vocabulary terms should be inspected. Ranking inconsistencies helps in reducing the effort required during the resolution of inconsistencies in both rules and vocabularies. The concrete steps are the following:

1. *Rules inconsistency detection* Inconsistencies in RML rules are detected via a rule-based reasoning system [12];
2. *RML rules clustering* RML rules are clustered according to the Triples Map to which the rules, i.e., Term Maps, correspond;
3. *RML rules ranking* A score for every rules cluster and vocabulary term is calculated by iterating over each cluster that represents an entity and every Terms Map in the cluster to count inconsistencies in which a rule is involved;
4. *RML rules refinement* experts inspect the top rules clusters, and apply the necessary refinements to the RML rules;

5. *Knowledge graph generation* The knowledge graph is generated by applying the semantic annotations to existing data sources via the RML rules;
6. *Knowledge graph inconsistency detection* The knowledge graph is validated to determine inconsistencies.
7. *RML rules refinement* Inconsistencies detected in the previous steps might be resolved by refining the RML rules and vocabulary terms.

5 Conclusions

In this paper, we discuss in detail the differences in syntax between the W3C recommended R2RML mapping language and its most broadly used generalisation RML. Both R2RML and RML may be used for generating RDF from data in relational databases. If a data owner only holds data in relational databases, using R2RML should suffice. However, if a dataowner holds data which is heterogeneous and not in relational databases, then RML is the only option.

While RML generalises R2RML, it also inherits R2RML's limitations. Thus, there are still aspects that are not supported by none of the two languages, such as RDF collections and containers or nested term maps. xR2RML [33], an extension over both R2RML and RML, is the only work which proposes a solution towards this direction, but more research is required.

As far as rules validation is concerned, more research is required to further investigate how rules can be validated and more importantly how inconsistencies may be resolved. Even though research is performed recently aiming to increase the quality of knowledge graphs, there is still room for improvement. Approaches applied to the rules that generate the knowledge graph point exactly to the root causing the inconsistency and prevent violations from being propagated. They also require significantly less time and resources, as opposed to time-consuming and performance-intensive approaches applied to the generated knowledge graph. Even though preliminary efforts propose methodologies to resolve inconsistencies, more research is still required to achieve more automated approaches that require less human intervention and contribute in generating higher quality knowledge graphs.

References

1. M. Acosta, A. Zaveri, E. Simperl, D. Kontokostas, S. Auer, and J. Lehmann. Crowdsourcing linked data quality assessment. In *The Semantic Web*, 2013.
2. C. B. Aranda, O. Corby, S. Das, L. Feigenbaum, P. Gearon, B. Glimm, S. Harris, S. Hawke, I. Herman, N. Humfrey, N. Michaelis, C. Ogbuji, M. Perry, A. Passant, A. Polleres, E. Prud'hommeaux, A. Seaborne, and G. T. Williams. SPARQL 1.1 Overview. Recommendation, World Wide Web Consortium (W3C), 2013.
3. M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. A Direct Mapping of Relational Data to RDF. Recommendation, World Wide Web Consortium (W3C), 2012.

4. C. Bizer and R. Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, 2009.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The mastro system for ontology-based data access. *Semantic Web*, 2011.
6. R. Catherine, B. Stephan, A. Géraldine, and B. Daniel. Weather data publication on the lod using sosa / ssn ontology. *Semantic Web*, 2019.
7. R. Cyganiak, C. Bizer, J. Garbers, O. Maresch, and C. Becker. The D2RQ Mapping Language. Technical report, FU Berlin, DERI, UCB, JP Morgan Chase, AGFA, HP Labs, Johannes Kepler Universitt Linz, 2012.
8. R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium (W3C), 2014.
9. S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. Working group recommendation, World Wide Web Consortium (W3C), 2012.
10. B. De Meester and A. Dimou. The Function Ontology. Unofficial Draft, Ghent University – imec – IDLab, 2016.
11. B. De Meester, A. Dimou, R. Verborgh, and E. Mannens. Detailed provenance capture of data processing. In D. Garijo, W. R. van Hage, T. Kauppinen, T. Kuhn, and J. Zhao, editors, *Proceedings of the First Workshop on Enabling Open Semantic Science (SemSci)*, volume 1931 of *CEUR Workshop Proceedings*, 2017.
12. B. De Meester, P. Heyvaert, D. Arndt, A. Dimou, and R. Verborgh. RDF Graph Validation Using Rule-Based Reasoning. *Semantic Web Journal*, 2020. Accepted.
13. B. De Meester, P. Heyvaert, R. Verborgh, and A. Dimou. Mapping languages analysis of comparative characteristics. In D. Chaves-Fraga, P. Heyvaert, F. Priyatna, J. Sequeda, A. Dimou, H. Jabeen, D. Graux, G. Sejdiu, M. Saleem, and J. Lehmann, editors, *Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics co-located with 16th Extended Semantic Web Conference (ESWC)*, volume 2489 of *CEUR Workshop Proceedings*, pages 37–45, Portorož, Slovenia, 2019.
14. 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 ESWC*, volume 10250 of *LNCS*. Springer, 2017.
15. A. Dimou, T. De Nies, R. Verborgh, E. Mannens, and R. Van de Walle. Automated metadata generation for Linked Data generation and publishing workflows. In S. Auer, T. Berners-Lee, C. Bizer, and T. Heath, editors, *Proceedings of the Workshop on Linked Data on the Web co-located with 25th International World Wide Web Conference (WWW2016)*, CEUR Workshop Proceedings, 2016.
16. A. Dimou, D. Kontokostas, M. Freudenberg, R. Verborgh, J. Lehmann, E. Mannens, S. Hellmann, and R. Van de Walle. Assessing and Refining Mappings to RDF to Improve Dataset Quality. pages 133–149, 2015.
17. A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proceedings*. CEUR, 2014.
18. A. Dimou, R. Verborgh, M. V. Sande, E. Mannens, and R. Van de Walle. Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval. In *Proceedings of the 11th International Conference on Semantic Systems*, SEMANTICS '15, pages 145–152. ACM, 2015.
19. M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). Standard track, IETF, 2005.

20. M. Farid, A. Roatis, I. F. Ilyas, H.-F. Hoffmann, and X. Chu. CLAMS: Bringing Quality to Data Lakes. In F. Özcan and G. Koutrika, editors, *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, pages 2089–2092, New York, United States, 2016. ACM.
21. P. Haase. Hybrid Enterprise Knowledge Graphs. Technical report, metaphacts GmbH, 2019.
22. P. Heyvaert, B. De Meester, A. Dimou, and R. Verborgh. Declarative Rules for Linked Data Generation at your Fingertips! 2018.
23. P. Heyvaert, A. Dimou, B. De Meester, and R. Verborgh. Rule-driven inconsistency resolution for knowledge graph generation rules. *Semantic Web Journal*, 10(6):1071–1086, 2019.
24. A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker. An Empirical Survey of Linked Data Conformance. *Journal of Web Semantics*, 2012.
25. A. C. Junior, C. Debruyne, R. Brennan, and D. OSullivan. 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*, page 267275, New York, USA, 2016. ACM.
26. D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven evaluation of linked data quality. In C.-W. Chung, editor, *Proceedings of the 23rd international conference on World Wide Web*, pages 747–757, New York, United States, 2014. ACM.
27. C. Lange. Krextor - An Extensible Framework for Contributing Content Math to the Web of Data. In J. H. Davenport, W. M. Farmer, J. Urban, and F. Rabe, editors, *Intelligent Computer Mathematics*, pages 304–306. Springer, 2011.
28. A. Langegger and W. Wöß. XLWrap – Querying and Integrating Arbitrary Spreadsheets with SPARQL. In *The Semantic Web - ISWC 2009*. Springer, 2009.
29. M. Lefrançois, A. Zimmermann, and N. Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *The Semantic Web 14th International Conference, ESWC 2017, Proceedings*, Portoroz, Slovenia, 2017. Springer.
30. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2), 2015.
31. E. Makela, E. Hyvönen, and T. Ruotsalo. How to deal with massively heterogeneous cultural heritage data: Lessons learned in culturesampo. *Semantic Web*, 2012.
32. F. Michel, L. Djimenou, C. Faron-Zucker, and J. Montagnat. Translation of Heterogeneous Databases into RDF, and Application to the Construction of a SKOS Taxonomical Reference. In *International Conference on Web Information Systems and Technologies*, pages 275–296. Springer, 2015.
33. F. Michel, L. Djimenou, C. Faron-Zucker, and J. Montagnat. xR2RML: Relational and Non-Relational Databases to RDF Mapping Language. Rapport de recherche, Laboratoire d'Informatique, Signaux et Systmes de Sophia-Antipolis (I3S), 2017.
34. P. F. Patel-Schneider. Using Description Logics for RDF Constraint Checking and Closed-World Recognition. In B. Bonet and S. Koenig, editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 247–253. AAAI Press, 2015.
35. H. Paulheim. Data-driven Joint Debugging of the DBpedia Mappings and Ontology. 2017.
36. F. Scharffe, G. Atemezing, R. Troncy, F. Gandon, S. Villata, B. Bucher, F. Hamdi, L. Bihanic, G. Képéklian, F. Cotton, J. Euzenat, Z. Fan, P.-Y. Vandenbussche, and B. Vatant. Enabling Linked Data publication with the Datalift platform. In *AAAI Workshop on Semantic Cities*, Toronto, ON, Canada, 2012.