

Conformance Test Cases for the RDF Mapping Language (RML)

Pieter Heyvaert¹[0000-0002-1583-5719], David
Chaves-Fraga²[0000-0003-3236-2789], Freddy Priyatna²[0000-0002-9292-4010],
Oscar Corcho²[0000-0002-9260-0753], Erik Mannens¹[0000-0001-7946-4884], Ruben
Verborgh¹[0000-0002-8596-222X], and Anastasia Dimou¹[0000-0003-2138-7972]

¹ IDLab, Dept. of Electronics and Information Systems, Ghent University – imec

² Ontology Engineering Group, Universidad Politécnica de Madrid

Abstract. Knowledge graphs are often generated using rules that apply semantic annotations to data sources. Software tools then execute these rules and generate or virtualize the corresponding RDF-based knowledge graph. RML is an extension of the W3C-recommended R2RML language, extending support from relational databases to other data sources, such as data in CSV, XML, and JSON format. As part of the R2RML standardization process, a set of test cases was created to assess tool conformance the specification. In this work, we generated an initial set of reusable test cases to assess RML conformance. These test cases are based on R2RML test cases and can be used by any tool, regardless of the programming language. We tested the conformance of two RML processors: the RMLMapper and CARML. The results show that the RMLMapper passes all CSV, XML, and JSON test cases, and most test cases for relational databases. CARML passes most CSV, XML, and JSON test cases regarding. Developers can determine the degree of conformance of their tools, and users determine based on conformance results to determine the most suitable tool for their use cases.

Keywords: RML · R2RML · test case

1 Introduction

Knowledge graphs are often generated based on rules that apply semantic annotations to raw or semi-structured data. For example, the DBpedia knowledge graph is generated by applying classes and predicates of the DBpedia ontology to Wikipedia [1]. Software tools execute these rules and generate corresponding RDF triples and quads [2], which materialize knowledge graphs. In the past, custom scripts prevailed, but lately, rule-driven tools emerged. Such tools distinguish the *rules* that define how RDF terms and triples are generated from the *tool* that executes those rules. R2RML [3] is the W3C-recommended language to define such rules for generating knowledge graphs from data in relational databases (RDBs). An R2RML processor is a tool that, given a set of R2RML rules and a relational database, generates an RDF dataset. Examples of R2RML

processors include Ultrawrap [4], Morph-RDB [5], Ontop [6], and XSPARQL [7]. A subset of them was included in the RDB2RDF Implementation Report [8] which lists their *conformance* to the R2RML specification. Conformance is assessed based on whether the correct knowledge graph is generated for a set of rules and certain relational database.

Given that R2RML is focused on relational databases only, extensions and adaptations were applied to account for other types of data sources. These include RML [9], XSPARQL [7], and xR2RML [10]. RML provides an extension of R2RML to support heterogeneous data sources, including different *formats* such as CSV, XML, JSON, and *access interfaces*, such as files and Web APIs. Various RML processors emerged, such as the RMLMapper³, CARML⁴, GeoTriples [11], and Ontario⁵. Unlike R2RML, there are no test cases available to determine the conformance to the RML specification. As a result, processors are either not tested or only tested with custom test cases, which do not necessarily assess every aspect of the specification. Consequently, no implementation report is available that allows comparing the different processors that generate knowledge graphs from heterogeneous data sources based on the conformance to the specification. This way, it is hard to determine the most suitable processor for a certain use case.

In this work, we introduce an initial set of RML test cases, which contains 297 test cases based on the 62 existing R2RML test cases. Instead of only considering relational databases as data sources, we also consider data in CSV, XML, and JSON format. Furthermore, we tested the conformance of the RMLMapper and CARML: every test case was executed by both processors and we noted whether the generated knowledge graph matches the expected one. The corresponding implementation report is available at <http://rml.io/implementation-report>. This helps determining which processor is the most suitable for a certain use case. For example, do users want a processor that supports the complete specification, or do they prefer a processor that does not support certain aspects of the specification, but executes the rules faster?

The test cases results shows that the RMLMapper (v4.3.2) passes all test cases regarding CSV, XML, and JSON format, and most test cases for RDBs, but fails the test cases for automatic datatyping of literals. CARML (v0.2.3) passes most test cases regarding CSV, XML, and JSON format, except of the test cases that deal, for example, with multiple RDF terms generation. Users can now determine how conformant the different processors are to the RML specification and use this conformance to determine the most suitable processor for their use cases.

The remainder of the paper is structured as follows. In Section 2, we discuss related work. In Section 3, we discuss the test cases. In Section 4, we elaborate on the test cases execution and results. In Section 5, we conclude the paper.

³ RMLMapper, <https://github.com/RMLio/rmlmapper-java>

⁴ CARML, <https://github.com/carm1/carm1>

⁵ Ontario, <https://github.com/WDAqua/Ontario>

2 Related work

In this section, we describe the related work that is relevant to the paper. First, we explain the most important knowledge graph generation language specifications, including R2RML and RML, and processors that execute those rules. Second, we discuss the differences between R2RML and RML. Finally, we describe the R2RML test cases, how they are defined and implemented and their corresponding implementation report with results of a few processors.

2.1 Knowledge graph generation languages and tools

R2RML [3] is the W3C recommended language for describing rules to generate RDF from data in RDBs. Currently, many tools support this specification. These tools follow either an Extract-Transform-Load (ETL) process, where a knowledge graph is materialized, e.g., DB2Triples⁶ and R2RMLParser⁷, or they provide virtual RDF views, focusing more on formalizing the translation from SPARQL to SQL and optimizing the resulted SQL query, e.g., Morph-RDB⁸ and Ontop⁹.

We describe in more details pioneering tools for executing R2RML rules: DB2Triples is a tool for extracting data from relational databases, semantically annotating the data extracts according to R2RML rules and generating Linked Data. The R2RMLParser [12] deals in principle with incremental Linked Data generation. Each time a knowledge graph is generated, not all data is used, but only the one that changed (so-called incremental transformation). Morph-RDB [5] and Ontop [6] adapt the algorithm defined by Chebotko, Lu, and Foutouhi [13] on SPARQL-to-SQL translation, using the information provided by the R2RML rules. Both apply several semantic optimizations (e.g., self join elimination) that generate efficient SQL queries to speed up the evaluation time.

RML [9] is defined as an extension of R2RML to specify rules for generating knowledge graphs from data in different formats, such as CSV, JSON, XML, and different access interfaces, e.g., open data connectivity and Web APIs [14]. Different other languages build upon RML for generating knowledge graphs from heterogeneous data sources, e.g., xR2RML [10] or RMLC [15].

A set of processors that support the RML specification are proposed. The RMLMapper is a Java library and command line interface that executes RML rules to generate RDF. Following the same approach, CARML executes RML rules, but also includes its own extensions, such as MultiTermMap (to deal with arrays) and XML namespace (to improve XPath expressions). GeoTriples [11] is a processor that generates and executes RML rules for generating RDF from geospatial data from different sources. The processor supports data stored in raw files (shapefiles, CSV, KML, GML, and so on), but also geospatial RDBs such as

⁶ DB2triples, <https://github.com/antidot/db2triples>

⁷ R2RMLParser, <https://github.com/nkons/r2rml-parser>

⁸ Morph-RDB, <https://github.com/oeg-upm/morph-rdb>

⁹ Ontop, <https://github.com/ontop/ontop>

	R2RML	RML
input reference	Logical Table	Logical Source
data source language	SQL (implicit)	Reference Formulation (explicit)
value reference	column	logical reference (valid expression following Reference Formulation)
iteration	per row (implicit)	per record (explicit – valid expression following Reference Formulation)

Table 1. Summary of the main differences between R2RML and RML

PostGIS¹⁰ and MonetDB¹¹. The generated RDF is based on well-known geospatial vocabularies, such as GeoSPARQL [16] and stSPARQL [17]. Ontario [18] is a federated query processor that uses RML rules to transform heterogeneous data sources during the query processing. Basically, the processor performs the generation using RML during the query processing step and executes federated SPARQL queries over the resulted RDF graphs. These processors are evaluated using ad-hoc examples or feasibility approaches, but a thorough representation of their capabilities is not provided. For that reason, we notice that RML test cases are needed to assess the capabilities of the different processors.

2.2 R2RML and RML differences

RML is an extension of R2RML and, thus, follows the core concepts of R2RML’s specification, such as Triples Maps, Term Maps, Subject Maps, and so on. However, there is a difference on the reference to the data to support heterogeneous data sources with respect to their format, e.g. CSV, XML, JSON, and access interface, e.g. files or Web APIs (see Table 1).

Logical Source A Logical Source extends R2RML’s Logical Table and describes the input data source used to generate the RDF. The Logical Table is only able to describe relational databases, whereas the Logical Source defines different heterogeneous data sources, including relational databases.

Reference Formulation As RML is designed to support heterogeneous data sources, data sources in different formats needs to be supported. One refers to data in a specific format according to the grammar of a certain formulation, which might be path and query languages or custom grammars. For example, one can refer to data in an XML file via XPath and in a relational database via SQL. To this end, the *Reference Formulation* was introduced indicating the formulation used to refer to data in a certain data source.

¹⁰ PostGIS, <https://postgis.net/>

¹¹ MonetDB, <https://www.monetdb.org/>

Iterator In R2RML it is specified that processors iterate over each row to generate RDF. However, as RML is designed to support heterogeneous data sources, the iteration pattern cannot always be implicitly assumed. For example, iterating over a specific set of objects is done by selecting them via a JSONPath expression. To this end, the *Iterator* was introduced which determines the iteration pattern over the data source and specifies the extract of data used to generate RDF during each iteration. The iterator is not required to be specified if there is no need to iterate over the input data.

Logical Reference When referring to values in a table or view of a relational database, R2RML relies on column names. However, as RML is designed to support heterogeneous data sources, rules may also refer to elements and objects, such as in the case of XML and JSON. Consequently, references to values should be valid with respect to the used reference formulation. For example, a reference to an attribute of a JSON object should be a valid JSONPath expression. To this end, (i) the `rml:reference` is introduced to replace `rr:column`, (ii) when a template is used, via `rr:template`, the values between the curly brackets should have an expression that is valid with respect to the used reference formulation, and (iii) `rr:parent` and `rr:child` of a Join Condition should also have an expression that is valid with respect to the used reference formulation.

2.3 W3C recommendations and their test cases

In the context of Semantic Web, several specifications were recommended by W3C, such as SPARQL [19], RDF [2], SHACL [20], Direct Mapping of relational data to RDF (DM) [21], and R2RML [3]. Each of these specifications has several related tools that support them. A set of test cases was defined for each one of them (SPARQL test cases¹², RDF 1.1 test cases¹³, SHACL test cases¹⁴, and R2RML and Direct Mapping test cases¹⁵, respectively) that provides useful information to choose the tool that fits better to certain needs. It is also a relevant step in the standardisation process of a technology or specification. We describe the R2RML in more details as it is related to the scope of this paper.

Determining the conformance of tools executing R2RML rules in the process of RDF generation is a step to provide objective information about the features of each tool. For this reason, the R2RML test cases [22] were proposed. It provides a set of 63 test cases. Each test case is identified by a set of features, such as the SQL statements to load the database, title, purpose, specification reference, review status, expected result, and corresponding R2RML rules. All the test cases are semantically described using the RDB2RDF-test¹⁶ and Test Metadata Vocabulary¹⁷. Several R2RML processors were assessed for their conformance with

¹² <https://www.w3.org/2001/sw/DataAccess/tests/r2>

¹³ <http://www.w3.org/TR/rdf11-testcases/>

¹⁴ <http://w3c.github.io/data-shapes/data-shapes-test-suite/>

¹⁵ <https://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/>

¹⁶ <http://purl.org/NET/rdb2rdf-test#>

¹⁷ <https://www.w3.org/TR/2005/NOTE-test-metadata-20050914/>

the R2RML specification running the test-cases. The results are available in the R2RML implementation-report [8]. The results are also annotated semantically using the Evaluation and Report Language (EARL) 1.0 Schema¹⁸.

3 RML test cases

In this section, we propose test cases to determine the conformance of RML processors to the RML specification. The proposed test cases are based on the R2RML test cases, but they take into account different heterogeneous data sources and the corresponding differences in RML (see Section 2.2). Our preliminary set of test cases includes (i) adjusted R2RML test cases for relational databases (including MySQL¹⁹, PostgreSQL²⁰, and SQL Server²¹) and (ii) new test cases for files in the CSV, XML (with XPath as the reference formulation), and JSON format (with JSONPath as the reference formulation). The test cases are described at <http://rml.io/test-cases/> and the corresponding files are available at <https://github.com/rmlio/rml-test-cases>. In Section 3.1, we describe the data model that is used to represent the test cases. In Section 3.2, we elaborate on the different files making up a test case. In Section 3.3, we discuss the differences between the R2RML and RML test cases.

3.1 Data model

We describe the test cases semantically to increase their reusability and shareability. To this end, we created a semantic data model²², with as main entity the test case (see Figure 1). For each test case, the following details are described: unique identifier, title, description, relevant aspect of the RML specification, data sources (optional), expected knowledge graph or error, and RML rules.

To provide the corresponding semantic descriptions, the model uses mostly the Evaluation and Report Language (EARL) 1.0 Schema²³, the Test case manifest vocabulary²⁴, the Test Metadata vocabulary²⁵, and the Data Catalog Vocabulary²⁶. A test case is annotated with the classes `earl:TestCase`, `test:TestCase`, and `mf:ManifestEntry`. The identifier, title, description, and the specific aspect of the RML specification that is being tested are added as datatype properties. The files that are provided as input to the tools are linked to the test cases via `test:informationResourceInput` and `dcterms:hasPart`. The file with the

¹⁸ <https://www.w3.org/TR/EARL10/>

¹⁹ <https://www.mysql.com/>

²⁰ <https://www.postgresql.org/>

²¹ <https://www.microsoft.com/en-us/sql-server/>

²² <http://rml.io/test-cases/#datamodel>

²³ <https://www.w3.org/TR/EARL10/>, with prefix `earl`

²⁴ <http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#>, with prefix `mf`

²⁵ <https://www.w3.org/2006/03/test-description#>, with prefix `test`

²⁶ <https://www.w3.org/TR/vocab-dcat/>, with prefix `dcat`

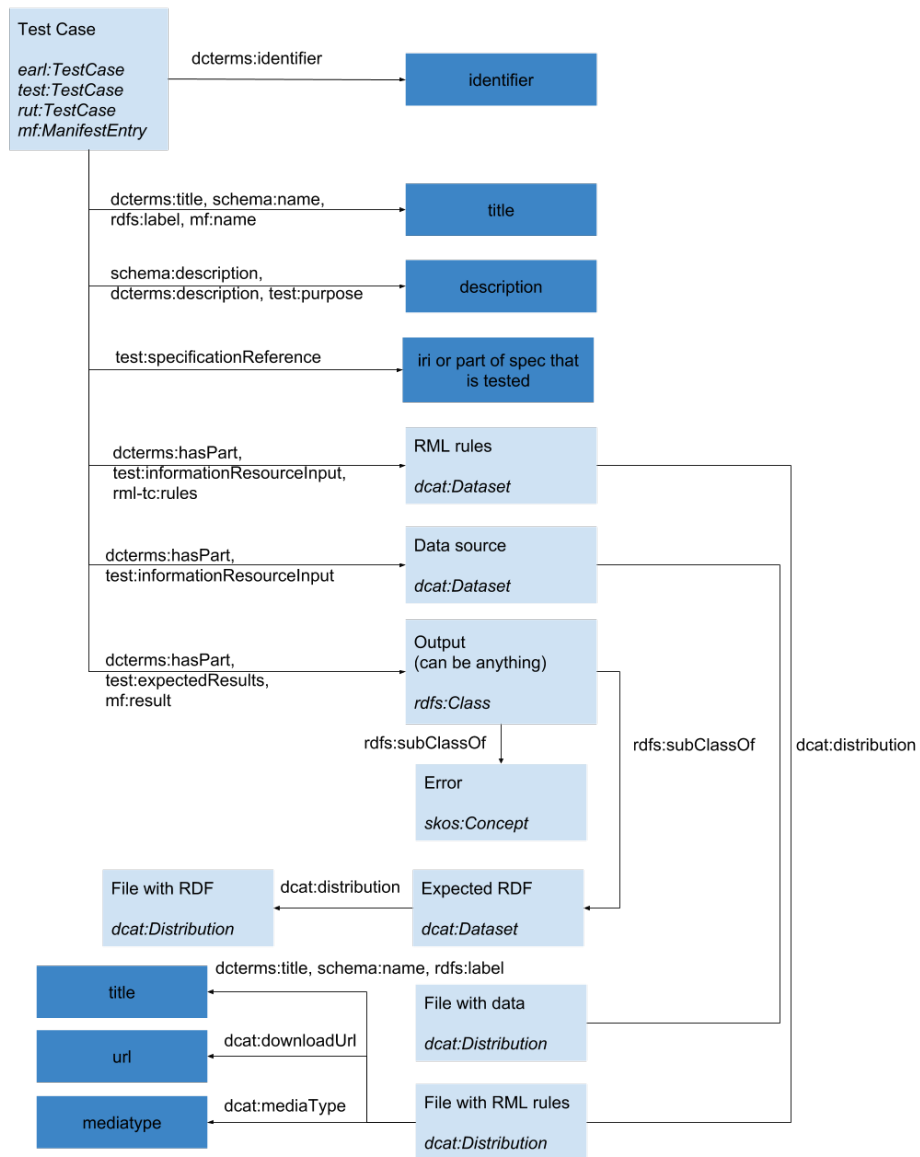


Fig. 1. Data model of the RML test cases

RML rules is also linked via `rml-tc:rules`²⁷. The objects of these properties are of the class `dcat:Dataset`, which in turn link to a `dcat:Distribution` that includes a link to a file. The expected output, whether that is a knowl-

²⁷ <http://rml.io/ns/test-cases>, with prefix `rml-tc`

edge graph or an error, is linked via `test:expectedResults`, `mf:result`, and `dcterms:hasPart`. In the case of a knowledge graph, the object of these properties is a `dcat:Dataset`, linked to a `dcat:Distribution`, to describe the file containing the graph. In the case of an error, we link to the expected error.

3.2 Test case files

Each test case consists of a set of files that contain the input data sources, the RML rules, and the expected RDF output. In practice, the files are organized as follows: all files for a single test case are contained in a single folder.

There are three types of files for each test case:

- 0 or more **data source files** for CSV (with extension `.csv`), XML (with extension `.xml`), and JSON (with extension `.json`), or 1 file with SQL statements to create the necessary tables for relational databases (called `resource.sql`);
- 1 **file with the RML rules** (in Turtle format, called `mapping.ttl`); and
- 0 or 1 **file with the expected RDF** (in N-Quads format, called `output.nq`).

Distinct test cases assess different behaviours of the processors. Certain test cases assess the behaviour of the tools when (i) the required data sources are not available, and others when (ii) an error occurs and no output is generated. In the former, no data sources files or SQL statements are provided. In the latter, no file with the expected RDF is provided. The test cases are independent of how the processors materialize the knowledge graph: a data dump, as done by the RMLMapper, or on the fly, as done by Ontario [23].

3.3 Differences with R2RML test cases

For most R2RML test cases, we created an RML variant for CSV, XML, JSON, MySQL, PostgreSQL, and SQL Server, leading to 6 RML test cases per R2RML test case. For R2RML test cases that focus on specific features of SQL queries, we only created 3 RML test cases, i.e., for MySQL, PostgreSQL, and SQL Server.

For test cases with CSV, XML, and JSON files as data sources, we created the corresponding files with the data based on the tables of the relational databases. For CSV, we used the table created by the SQL statements of the R2RML test case and stored it as a CSV file. For XML, the name of the table was used for the root of the XML document and every row of the table was used to create an XML element. Within this element, elements were created for each column and their values are the values of the corresponding columns in the table. For JSON, we followed a similar approach as XML. The file contains a JSON object at the root with the name of the table as the only attribute. This attribute has as value an array, where each element of the array corresponds with a row in the table. For each row, attributes were created for each column and their values are the values of the corresponding columns in the table.

Data errors. 2 of the R2RML test cases expect a data error to happen, e.g., when the subject IRI of an entity cannot be generated. In this case, an error is thrown and no knowledge graph is generated. With RML for entities where no subject IRI can be generated there is also no output generated, but, in contrast to R2RML, for the other entities the corresponding output is still generated. Therefore, for the corresponding RML test cases the processors can still throw an error, but the generation of the knowledge graph must not be halted.

Inverse expressions. 3 of the R2RML test cases are designed to test the use of inverse expressions²⁸. However, inverse expressions are only used to optimize the knowledge graph generation and no differences are observed in the generated knowledge graph. Thus, whether inverse expressions are used by a processor or not cannot be verified by such test cases. Thus, we do not include them for RML.

SQL-specific features. 18 of the R2RML test cases focus on specific features of SQL queries, e.g., a duplicate column name in a SELECT query. As there are no corresponding RML test cases for CSV files, XML files with XPath, and JSON files with JSONPath, we only provide 54 corresponding test cases for MySQL, PostgreSQL, and SQL Server.

Null values. 1 of the R2RML test cases tests null values in the rows. However, a corresponding RML test case cannot be provided for the CSV and XML format, because both formats do not support null values.

Spaces in columns. 1 of the R2RML test cases is designed to test the behaviour when dealing with spaces in the columns of the SQL tables. However, a corresponding RML test case cannot be provided for the XML format, because it does not allow spaces in names.

In total, we have 297 test cases: 39 for CSV, 38 for XML, 40 for JSON, and 180 for relational databases. Of these 297, 255 test cases expect an knowledge graph to be generated, while 36 expect an error that halts the generation.

4 Test case execution and results

In this section, we describe the execution of the test cases and their results for two RML processors: the RMLMapper (v4.3.2) and CARML (v0.2.3). The implementation report can be found at <http://rml.io/implementation-report>.

We ran the RML test cases over the RML processors and annotated the obtained results using the EARL Schema. Three types of results are possible per test case: “passed”, “failed”, and “inapplicable”. **Passed** (`earl:passed`) is used either when the actual output matches the expected output when no error is expected, or when the tool throws an error when an error is expected. **Failed**

²⁸ <https://www.w3.org/TR/r2rml/#inverse>

(`earl:failed`) is used either when the actual output does not match the expected output if no error is expected, the processor returns an error trying to execute a test or the tool does not throw error if an error is expected. **Inapplicable** (`earl:inapplicable`) is used when the tool clearly states that specific features are not supported.

	CSV	XML	JSON	My-SQL	Post-gres	SQL Server	total
passed	39	38	40	53	54	53	277
failed	0	0	0	7	6	7	20
inapplicable	0	0	0	0	0	0	0

Table 2. The RMLMapper (v4.3.2) passes the majority of test cases, with the exception of 20 cases for relational databases.

	CSV	XML	JSON	My-SQL	Post-gres	SQL Server	total
passed	29	28	27	0	0	0	84
failed	10	10	13	0	0	0	33
inapplicable	0	0	0	60	60	60	180

Table 3. CARML (v0.2.3) passes almost 3 out of 4 test cases, but does not support relational databases.

In Table 2 we show the results for the RMLMapper processor. It passes all CSV, JSON and XML test cases, but fails in 20 test cases for the RDBs. The failures are related to the automatic datatyping of literal for RDBs specified by R2RML²⁹. RMLMapper should pass the failed test cases in next versions of the processor.

In Table 3 we show the results for the CARML processor. It partially passes the CSV, JSON and XML test cases, but it does not provide support for any of the RDBs test cases. The failures are related to the unsupported for multiple Subject Maps, multiple Predicate Maps, and Named Graphs. The developers of the tool declare that CARML will support these features in next versions of the processor. However, at the moment of writing, we do not have any information about whether CARML will provide support for RDBs.

Finally, we can declare that testing a RML processor with the defined cases and analysing the obtained results offers a general view of the current status of it. These results also give useful information to the tool developers on knowing where they should put their effort to improve the conformance of the processor.

²⁹ <https://www.w3.org/TR/r2rml/#dfn-natural-rdf-literal>

5 Conclusion

With the introduction of an initial set of RML test cases (i) developers can determine how conformant their RML processors are to the RML specification, and (ii) users can use the test cases results to select the most appropriate processor for a specific use case. Before, users were only able to rely on the custom test cases, if any, which not necessarily assess every aspect of the specification. Now, users can rely on well-defined set of test cases that (a) clearly define what the input and expected output is, and (b) are reusable across different processors written in different programming languages.

The results of the test cases execution with the RMLMapper and CARML show that the CSV, XML, and JSON formats are almost fully supported, but RDBs cause difficulties or are not supported at all. The RMLMapper passes more test cases than CARML and therefore, the former is better when considering the conformance to the RML specification.

Our set of test cases is based on the R2RML test cases, and therefore, it covers a big part of the RML specification, as it is based on R2RML. However, as the R2RML test cases focus on relational databases, they do not take into account the specifics of hierarchical data formats, such as nested structures in JSON and XML files, which can be used with RML. Therefore, further research should be directed towards creating new test cases that tackle these specifics taking into account the differences between the different hierarchical formats and their corresponding reference formulations.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union. The work presented in this paper is partially supported by the Spanish Ministerio de Economía, Industria y Competitividad and EU FEDER funds under the DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish national project (TIN2016-78011-C4-4-R) and by an FPI grant (BES-2017-082511).

References

- [1] Jens Lehmann et al. “DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia”. In: *Semantic Web 6* (2015), pp. 167–195.
- [2] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. Recommendation. World Wide Web Consortium (W3C), Feb. 2014. URL: <http://www.w3.org/TR/rdf11-concepts/>.
- [3] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. <http://www.w3.org/TR/r2rml/>. W3C, 2012.

- [4] Juan F. Sequeda and Daniel P. Miranker. “Ultrawrap: SPARQL execution on relational data”. In: *Web Semantics: Science, Services and Agents on the WWW* (2013). ISSN: 1570-8268. DOI: <https://doi.org/10.1016/j.websem.2013.08.002>. URL: <http://www.sciencedirect.com/science/article/pii/S1570826813000383>.
- [5] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. “Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph”. In: *23rd International Conference on WWW*. 2014. ISBN: 978-1-4503-2744-2.
- [6] Diego Calvanese et al. “Ontop: Answering SPARQL Queries over Relational Databases”. In: *Semantic Web Journal* (2017).
- [7] Stefan Bischof et al. “Mapping between RDF and XML with XSPARQL”. In: *Journal on Data Semantics* (2012). ISSN: 1861-2040. DOI: 10.1007/s13740-012-0008-7. URL: <http://dx.doi.org/10.1007/s13740-012-0008-7>.
- [8] Boris Villazón-Terrazas and Michael Hausenblas. *RDB2RDF Implementation Report*. W3C Note. <https://www.w3.org/TR/rdb2rdf-implementations/>. W3C, 2012.
- [9] Anastasia Dimou et al. “RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data”. In: *LDOW*. 2014.
- [10] Franck Michel et al. “Translation of Relational and Non-relational Databases into RDF with xR2RML”. In: *WEBIST*. 2015.
- [11] Kostis Kyzirakos et al. “GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings”. In: *Journal of Web Semantics* 52 (2018), pp. 16–32.
- [12] Nikolaos Konstantinou et al. “Exposing scholarly information as Linked Open Data: RDFizing DSpace contents”. In: *The Electronic Library* 32.6 (2014), pp. 834–851. DOI: 10.1108/EL-12-2012-0156.
- [13] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. “Semantics preserving SPARQL-to-SQL translation”. In: *Data & Knowledge Engineering* 68.10 (2009), pp. 973–1000.
- [14] Anastasia Dimou et al. “Machine-interpretable Dataset and Service Descriptions for Heterogeneous Data Access and Retrieval”. In: *Proceedings of the 11th International Conference on Semantic Systems*. SEMANTICS ’15. ACM, 2015. ISBN: 978-1-4503-3462-4. DOI: 10.1145/2814864.2814873. URL: <http://doi.acm.org/10.1145/2814864.2814873>.
- [15] David Chaves-Fraga et al. “Virtual Statistics Knowledge Graph Generation from CSV files”. In: *Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events*. Vol. 36. Studies on the Semantic Web. IOS Press, 2018, pp. 235–244.
- [16] Robert Battle and Dave Kolas. “Geosparql: enabling a geospatial semantic web”. In: *Semantic Web Journal* 3.4 (2011), pp. 355–370.
- [17] Manolis Koubarakis and Kostis Kyzirakos. “Modeling and querying meta-data in the semantic sensor web: The model stRDF and the query language

- stSPARQL”. In: *Extended Semantic Web Conference*. Springer, 2010, pp. 425–439.
- [18] Maria-Esther Vidal et al. “Semantic Data Integration of Big Biomedical Data for Supporting Personalised Medicine”. In: *Current Trends in Semantic Web Technologies: Theory and Practice*. Springer, 2019, pp. 25–56.
 - [19] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. Recommendation. World Wide Web Consortium (W3C), Mar. 2013. URL: <https://www.w3.org/TR/sparql11-query/>.
 - [20] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. Recommendation. World Wide Web Consortium (W3C), 2017. URL: <https://www.w3.org/TR/shacl/>.
 - [21] Marcelo Arenas et al. *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. <https://www.w3.org/TR/rdb-direct-mapping/>. W3C, Sept. 2012.
 - [22] Boris Villazón-Terrazas and Michael Hausenblas. *R2RML and Direct Mapping Test Cases*. W3C Note. <http://www.w3.org/TR/rdb2rdf-test-cases/>. W3C, 2012.
 - [23] Anastasia Dimou et al. “What Factors Influence the Design of a Linked Data Generation Algorithm?” In: *Proceedings of the 11th Workshop on Linked Data on the Web*. Ed. by Tim Berners-Lee et al. Apr. 2018. URL: http://events.linkedata.org/ldow2018/papers/LDOW2018_paper_12.pdf.