Backwards or Forwards? [R2]RML Backwards Compatibility in RMLMapper

Dylan Van Assche[1,*], Jozef Jankaj[1] and Ben De Meester[1,*]

[1]IDLab, Dept. Electronics & Information Systems, Ghent University – imec, Belgium

Abstract

During the past decade, RML was proposed as an extension to the W3C's R2RML Recommendation for supporting heterogeneous data sources. Although RML (RML_{io} flavour) was not a W3C Recommendation, it gained a lot of traction, and has been extended by the KG-Construct W3C Community Group as RML_{KGC}. Currently, this results in three main flavours (i.e. R2RML, RML_{io}, and KG-Construct's RML_{KGC}) used among users of these mapping languages. Therefore, many existing mappings cannot be used among all existing [R2]RML engines, since they only implement one [R2]RML flavour. In this paper, we implement a translation of all flavours into the latest RML flavour (i.e. RML_{KGC}) within RMLMapper. This way, any mapping – no matter which flavour of [R2]RML was used – can be executed by RMLMapper. We discuss our translation approach and evaluate it in the KGCW Challenge 2024 Track 1 and all available RML_{io} and R2RML test cases to verify our translation into RML_{KGC}. We were able to translate R2RML and RML_{io} to RML_{KGC} Core (98,7%) and some parts of RML_{KGC} IO (50,75%) modules without changing the [R2]RML mappings. We reach a total coverage of 73,70% among all RML_{KGC} test cases and 100% coverage for RML_{io} and R2RML test cases. Thanks to our translation approach, we can re-use the same RMLMapper for all flavours without requiring the user to change their mappings. In the future, we aim to support all RML_{KGC} modules, while keeping support for the other flavours.

Keywords

RML, Knowledge Graph Construction, RMLMapper, Challenge

1. Introduction

During the past decade, RML was proposed as an extension to the W3C's R2RML Recommendation for supporting heterogeneous data sources. On its own, RML has been revised by the KG-Construct W3C Community Group.

Nowadays, multiple flavours of [R2]RML exist: W3C's Recommended R2RML specification [1] (R2RML), the RML specification initiated by Dimou [2] (RML_{io}) and maintained throughout the years on https://rml.io (RML_{io}, v1.1.2[1]), and a new major revision [3] (RML_{KGC}), maintained by the W3C Community Group on Knowledge Graph Construction (RML_{KGC}, by KG-Construct). Not all flavours are supported by existing RML engines [4] such as SDM-RDFizer [5], Morph-KGC [6], RMLMapper [7], RMLStreamer [8].

KGCW'24: 5th International Workshop on Knowledge Graph Construction, May 27, 2024, Crete, GRE

*Corresponding author.

dylan.vanassche@ugent.be (D. Van Assche); jozef.jankaj@ugent.be (J. Jankaj); ben.demeester@ugent.be (B. De Meester)

https://dylanvanassche.be (D. Van Assche); https://ben.de-meester.org/#me (B. De Meester)

0000-0002-7195-9935 (D. Van Assche); 0000-0003-0248-0987 (B. De Meester)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]https://rml.io/specs/rml/v/1.1.2/

Users with existing RML mappings are limited in the RML engines they can use, since no engine supports all [R2]RML flavours. RML_{KGC} can represent all elements present in R2RML and RML_{io} through translation because each specification is backwards compatible with each other. However, no engine takes advantage of this backwards compatibility to represent the other flavours as RML_{KGC}. Therefore, users must translate all their existing mappings first if they want to use a different engine which supports a newer flavour. Moreover, we do not want to deprecate support for R2RML and RML_{io} flavours in our existing mapping engine RMLMapper[2] when adding support for RML_{KGC}. We overcome this problem in RMLMapper by translating all three [R2]RML flavours into RML_{KGC} [3]. By applying our translation, RMLMapper can now read any RML mapping without requiring the user to change them, written in R2RML, RML_{io}, and RML_{KGC}.

Thanks to our translation in RMLMapper, users can still run their existing [R2]RML mappings while the Knowledge Graph Construction community can work towards the standardization of RML as a W3C Recommendation in the future.

2. Approach

In this Section, we show our translation for R2RML and RML_{io} into RML_{KGC}. This translation is implemented in RMLMapper and automatically applied without any intervention of the user. Therefore, the user does not have to migrate existing [R2]RML mappings into RML_{KGC} immediately.

2.1. Translation

We compare the different [R2]RML flavours among each other to establish a translation path towards RML_{KGC}. Tables 1 & 2 list all required translations to translate R2RML and RML_{io} into RML_{KGC}. The biggest translations rely on the removal of R2RML Logical Table in favor of RML_{KGC} Logical Source, the ontology prefixes which are different between flavours, and the access descriptions for data sources. In this work, we cover the RML_{KGC} Core which focus on RDF generation with RML Triples Maps and parts of the RML_{KGC} IO modules used for accessing data sources and targets in RML because they overlap with the R2RML and RML_{io} flavours. Translation is required to avoid implementing all flavours separately in engines such as RMLMapper. In the future, we will expand our work to the other RML_{KGC} modules.

Prefixes Every [R2]RML flavour has its own prefix which allows us to recognize which flavour of [R2]RML was used to create the mapping. Since most of the terms in the ontologies are similar to each other, we translate the prefixes of R2RML and RML_{io} into RML_{KGC} by replacing them with the new prefix. However, some changes in RML_{KGC} require additional transformations which we describe in the next paragraphs.

Literals in `rml:source` RML_{io} mappings consistently use Literals in the RML_{io} Logical Source's `rml:source` to describe the path to a file which is used in the mapping as data source.

[2] https://github.com/RMLio/rmlmapper-java

Table 1
Translations from RML_{io} to RML_{KGC}. Queries used when accessing SQL databases and SPARQL endpoints need to be transformed into an iterator and corresponding reference formulation. File paths as string Literals in a RML Source must be transformed into a DCAT access description or RML_{KGC} Relative Path Source for relative file paths.

RML_{io}	**RML_{KGC}**
Classes	
`ql:XPath`	`rml:XPath`
`ql:CSV`	`rml:CSV`
`ql:JSONPath`	`rml:JSONPath`
`rml:LogicalSource`	`rml:LogicalSource`
`rml:BaseSource`	`rml:LogicalSource`
`rml:LanguageMap`	`rml:LanguageMap`
Properties	
`rml:iterator`	`rml:iterator`
`rml:logicalSource`	`rml:logicalSource`
`rml:reference`	`rml:reference`
`rml:referenceFormulation`	`rml:referenceFormulation`
`rml:languageMap`	`rml:languageMap`
Transformations	
`rml:query`	`rml:iterator` + `rml:referenceFormulation`
Literals in `rml:source`	DCAT or RML_{KGC} Relative Path Source

This approach was deprecated in 2015 [9] and replaced by access descriptions such as DCAT [10], SD [11], or D2RQ [12] to access heterogeneous data sources, e.g. files, SPARQL services, or databases. RML_{KGC} drops this deprecated option which requires a transformation when a mapping still uses Literals for `rml:source`. If we encounter such a case, we replace it by a DCAT access description. If the path to the file is a relative path, we cannot use DCAT since there is no base IRI available to resolve the relative path against. To overcome this problem, an RML_{KGC} Relative Path Source was introduced to handle this case[3].

`rml:query` The RML_{io} specification[4] does not indicate how queries must be specified in the case of relational databases or SPARQL services. Over the past decade, an unofficial predicate `rml:query` was used to address this problem. This way, queries could be specified in the RML_{io} mapping to access such sources. The W3C Community Group on Knowledge Graph Construction incorporated the query property in the iterator, but there is still discussion around this approach[5]. In this work, we do perform this transformation and add the necessary access descriptions for the relational database or SPARQL service if needed.

rr:tableName & rr:LogicalTable R2RML Logical Table and `rr:tableName` shortcut must be translated completely into RML_{KGC} Logical Source since a Logical Source is an expansion of an R2RML Logical Table. We perform this transformation by moving the query from R2RML

[3]https://github.com/kg-construct/rml-io/issues/36
[4]https://rml.io/specs/rml/v/1.1.2/
[5]https://github.com/kg-construct/rml-io/issues/28

Logical Table into the iterator and adding the access description of the database using the D2RQ ontology[6]. The reference formulation is set to `rml:SQL2008Table`. For `rr:tableName`, we also add the access description using the D2RQ ontology, and place the table name as well in the iterator. However, the reference formulation is set to `rml:SQL2008Table`, allowing RML engines to detect that they receive a table name instead of a SQL query.

2.2. Implementation in RMLMapper

Our translation is applied when RMLMapper parses the [R2]RML mappings. Each part of the [R2]RML mapping which is not using RML_{KGC}, is translated internally. This way, the RMLMapper operates on [R2]RML mappings based on the latest RML_{KGC} version. Only for R2RML, the database details needs to be supplied by the user as R2RML does not include the database access information in its mappings. Our implementation in the RMLMapper is written in Java, released as v7.0.0, and available on GitHub[7] under the MIT license.

3. Evaluation

In this Section, we evaluate our translation approach on the RML_{KGC} test cases of the Knowledge Graph Construction Workshop (KGCW) Challenge 2024 Track 1[8] to verify our implementation and identify which parts of the RML_{KGC} modules are (not) supported. Moreover, we have validated all R2RML and RML_{io} test cases' RDF output on the RMLMapper for correctness to avoid that our translation approach breaks the other [R2]RML flavors when translating into RML_{KGC}.

The KGCW Challenge 2024 Track 1 consists of 365 test cases from 5 different RML_{KGC} modules: RML_{KGC} Core (238 test cases), RML_{KGC} IO (67 test cases), RML_{KGC} FNML (13 test cases), RML_{KGC} CC (29 test cases), and RML_{KGC} Star (18 test cases). Each module provides a set of test cases to evaluate the compliance of engines with the specification provided by the module. RML_{KGC} Core has the most test cases because it contains the core functionality for generating RDF using RML_{KGC} mappings, followed by RML_{KGC} IO which focus on accessing various data sources and targets used in RML_{KGC} mappings. The other modules have lower number of test cases, thus engines supporting RML_{KGC} Core and RML_{KGC} IO already have a high coverage of the new RML_{KGC} flavour. We calculate the coverage of each module by dividing the number of passing test cases by the number of test cases per module. The Knowledge Graph Construction W3C Community Group does not provide a detailed description for each test case yet, but it is planned for the future.

Without our translation approach, RMLMapper achieves 0% coverage on the RML_{KGC} test cases of the KGCW Challenge. RMLMapper passes 100% of the R2RML and RML_{io} test cases. We achieve 98,7% coverage for RML_{KGC} Core and 50,75% coverage for RML_{KGC} IO. RML_{KGC} Core has a few test cases where RMLMapper fails to provide the correct output: For RML_{KGC} Core, the test case RMLTC0010{a,b,c}-JSON fails for RMLMapper as it uses the latest IETF JSONPath

[6] http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#
[7] https://doi.org/10.5281/zenodo.11518178
[8] https://doi.org/10.5281/zenodo.10721874

Table 2
Translations from W3C Recommended R2RML into RML_{KGC}. R2RML's specific access descriptions for SQL tables (Logical Table, BaseTableOrView, R2RMLView, tableName) need to be transformed into a RML_{KGC} Logical Source with a D2RQ Database access description for accessing SQL tables.

R2RML	RML_{KGC}
Classes	
rr:Literal	rml:Literal
rr:BlankNode	rml:BlankNode
rr:IRI	rml:IRI
rr:SQL2008	rml:SQL2008Table or rml:SQL2008Query
rr:TriplesMap	rml:TriplesMap
rr:SubjectMap	rml:SubjectMap
rr:PredicateObjectMap	rml:PredicateObjectMap
rr:PredicateMap	rml:PredicateMap
rr:ObjectMap	rml:ObjectMap
rr:TermMap	rml:TermMap
rr:GraphMap	rml:GraphMap
rr:Join	rml:Join
rr:RefObjectMap	rml:RefObjectMap
rr:defaultGraph	rml:defaultGraph
Properties	
rr:joinCondition	rml:joinCondition
rr:parent	rml:parent
rr:child	rml:child
rr:parentTriplesMap	rml:parentTriplesMap
rr:column	rml:reference
rr:class	rml:class
rr:constant	rml:constant
rr:datatype	rml:datatype
rr:graph	rml:graph
rr:graphMap	rml:graphMap
rr:language	rml:language
rr:object	rml:object
rr:objectMap	rml:objectMap
rr:predicate	rml:predicate
rr:predicateMap	rml:predicateMap
rr:predicateObjectMap	rml:predicateObjectMap
rr:subject	rml:subject
rr:subjectMap	rml:subjectMap
rr:termType	rml:termType
rr:template	rml:template
rr:logicalTable	rml:logicalSource
Transformations	
rr:BaseTableOrView	RML_{KGC} Logical Source + D2RQ Database
rr:R2RMLView	RML_{KGC} Logical Source + D2RQ Database
rr:Logical Table	RML_{KGC} Logical Source + D2RQ Database
rr:tableName	RML_{KGC} Logical Source + D2RQ Database

expressions which are not supported yet by the RMLMapper. For RML_{KGC} IO, new serialization and compression formats are not implemented in RMLMapper. Moreover, compressed data sources cannot be accessed yet by RMLMapper. We did not implement specific translations yet for FnO functions, provided by the RML_{KGC} FNML and other modules such as RML_{KGC} Star for RDF-Star support or RML_{KGC} CC to generate RDFS Collections & Containers. Therefore, we expected only a small set of test cases would succeed for these modules. The total coverage of RML_{KGC} test cases we reach for RMLMapper is 73,70%. RMLMapper still passes 100% of the R2RML and RML_{io} test cases with our translation approach.

4. Conclusion

In this paper, we showed our approach for translating R2RML and RML_{io} into the latest RML_{KGC} and evaluated it on the RML_{KGC} test cases. Thanks to our work, users can still execute their existing RML mappings while the community works towards a standardization of RML_{KGC} as a W3C Recommendation. In the future, we aim to support more RML_{KGC} modules besides RML_{KGC} Core and IO, and perform an evaluation of the translation itself since we focus in this work on participating in the KGCW Challenge which only evaluate the generated RDF of each engine.

Acknowledgments

The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Dylan Van Assche is supported by the Special Research Fund of Ghent University[9] under grant BOF20/DOC/132.

References

[1] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, Working Group Recommendation, World Wide Web Consortium (W3C), 2012. URL: http://www.w3.org/TR/r2rml/.

[2] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: C. Bizer, T. Heath, S. Auer, T. Berners-Lee (Eds.), Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184 of *CEUR Workshop Proceedings*, CEUR, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[3] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF, in: Submited to ISWC2023, 2023.

[9] https://www.ugent.be/en/research/funding/bof/overview.htm

[4] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, Journal of Web Semantics (2022). doi:10.1016/j.websem.2022.100753.

[5] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, M.-E. Vidal, SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, ACM, 2020. doi:10.1145/3340531.3412881.

[6] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, Semantic Web (2022) 1–20. doi:10.3233/sw-223135.

[7] P. Heyvaert, B. De Meester, D. Van Assche, et al., Rmlmapper, 2024. URL: https://github.com/RMLio/rmlmapper-java.

[8] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh, A. Dimou, Parallel RDF generation from heterogeneous big data, in: S. Groppe, L. Gruenwald (Eds.), Proceedings of the International Workshop on Semantic Big Data - SBD '19, number 1 in SBD '19, ACM Press, Amsterdam, Netherlands, 2019. URL: https://biblio.ugent.be/publication/8619808/file/8659668.pdf. doi:10.1145/3323878.3325802.

[9] A. Dimou, R. Verborgh, M. V. Sande, E. Mannens, R. V. 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, ACM Press, 2015. doi:10.1145/2814864.2814873.

[10] R. Albertoni, D. Browning, S. Cox, A. Gonzalez Beltran, A. Perego, P. Winstanley, Data Catalog Vocabulary (DCAT) - Version 2, Recommendation, World Wide Web Consortium (W3C), 2020. URL: https://www.w3.org/TR/vocab-dcat/.

[11] G. Williams, SPARQL 1.1 Service Description, Recommendation, World Wide Web Consortium (W3C), 2013. URL: https://www.w3.org/TR/sparql11-service-description/.

[12] R. Cyganiak, C. Bizer, J. Garbers, O. Maresch, C. Becker, The D2RQ Mapping Language, Technical Report, FU Berlin, DERI, UCB, JP Morgan Chase, AGFA Healthcare, HP Labs, Johannes Kepler Universität Linz, 2012. URL: http://d2rq.org/d2rq-language.