BURPing Through RML Test Cases

Dylan Van Assche[1,*,†], Christophe Debruyne[2,*,†]

[1]IDLab, Dept. Electronics & Information Systems, Ghent University – imec, Belgium
[2]Montefiore Institute, University of Liège, Belgium

Abstract

Recently, the W3C Community Group on Knowledge Graph Construction created a suite of test cases for all RML modules developed in the Community Group to verify implementations' compliance with the new RML specifications. However, these RML test cases could not be tested because no existing RML Processor supports them. In this paper, we report on our process of testing the new RML test cases while at the same time implementing support for the new RML modules in a reference implementation, which we call 'BURP' (Basic and Unassuming RML Processor), to investigate the feasibility and possible mistakes of the new RML test cases and specifications. We found several problems in the RML modules, ranging from mismatches between the test cases and their specification and invalid SHACL shapes to edge cases not covered by the specifications. Through this work, we improve the quality of RML test cases and the coverage of their corresponding specifications to increase adoption and conformance among RML Processors.

GitHub: https://github.com/kg-construct/BURP
DOI: https://zenodo.org/doi/10.5281/zenodo.11037711

Keywords

RML, RML Conformance Checking, Knowledge Graph Generation

1. Introduction

The W3C Community Group on Knowledge Graph Construction developed a suite of test cases to ensure implementations comply with the new RML specifications [1]. These test cases serve as a crucial resource for guaranteeing consistent behavior among RML Processors and the interoperability of RML mappings. Currently, no existing RML Processor can definitively verify the correctness of the test cases and the (interactions between the) various modules of the new RML specification. This is due to RML Processors [2] implementing the original RML specification [3], each focusing on additional functionalities and various optimizations for specific use cases (e.g., parallel processing)–these specificities impact engines' algorithms, so there is no reference implementation.

The problem is that no existing reference implementation for RML exists. This lack of a reference algorithm results in inconsistencies between test cases, specifications, and RML

Fifth International Workshop On Knowledge Graph Construction Co-located with the ESWC 2024, May 26-27 2024, Hersonissos, Greece

*Corresponding author.
†These authors contributed equally.
dylan.vanassche@ugent.be (D. Van Assche); c.debruyne@uliege.be (C. Debruyne)
https://dylanvanassche.be (D. Van Assche); http://christophedebruyne.be/ (C. Debruyne)
0000-0002-7195-9935 (D. Van Assche); 0000-0003-4734-3847 (C. Debruyne)

Processors, which hinders the community's adoption of the new specifications.

In this work, we introduce BURP (*Basic and Unassuming RML Processor*) to address this problem by providing a reference implementation on which the community can rely to develop new RML specifications, verify resources such as test cases and SHACL shapes, and compare implementations. BURP's development was driven by the need to comply with and sanity check test cases across specifications for the Knowledge Graph Construction Workshop Challenge.[1] BURP aims to become a reference implementation for all new RML modules. BURP has been designed to keep things simple: all data is processed in main memory using simple data structures. BURP does not rewrite mappings to optimize RDF generation. BURP's development has been instrumental in identifying and rectifying inconsistencies within the RML modules. This paper is structured as follows: Section 2 introduces our reference implementation BURP. Section 3 discusses all encountered problems in the RML test cases. Section 4 concludes this paper and discusses future work.

2. BURP

In this section, we introduce our reference RML implementation, the *Basic and Unassuming RML Processor* (BURP), which we developed to investigate the feasibility and consistency of the new RML specifications. We developed BURP from scratch to ensure we encounter all possible problems when implementing the RML modules and avoid any influence (i.e., assumptions) from existing RML Processors when establishing a reference implementation with a naive RDF generation algorithm.

BURP is *basic* since it uses simple data structures and an arguably naive approach to generate RDF, *unassuming* as no optimizations are applied when implementing the RML specifications. BURP follows simple steps to generate RDF, similar to the R2RML [4] reference algorithm. The code and RDF generation algorithm is deliberately kept simple to help RML Processors' developers implement the new RML specifications. Moreover, no attempts are made to optimize mappings (e.g., via mapping rewriting) or the process via distributing computing techniques to focus only on the actual RML specifications. BURP uses simple data structures that store all data in memory. BURP does not try to recover from or correct errors; BURP merely exits with a non-zero exit code when an error occurs. BURP will not even try to generate partial results, a feature arguably desirable in industry settings, as one only needs to rerun the failed mappings.

BURP currently fully supports *RML-Core*, *RML-CC*, and *RML-FNML*. It also supports some of the functionality of *RML-IO* (RML Logical Sources are supported, and the RML Logical Target is under development).

In the future, we aim to support all RML modules. BURP is written in Java, relying upon Apache's Jena Semantic Framework for managing the generated RDF, and is available on GitHub under the MIT license. Therefore, this implementation aims to serve as the community's sandbox for developing new RML modules and verifying their resources, e.g., RML mappings, SHACL shapes, test cases, etc.

[1] https://w3id.org/kg-construct/2024/challenge.html

3. Refining Test Cases

In this Section, we discuss (i) the RML test cases of the different RML modules and (ii) how we discovered problems, edge cases, and inconsistencies between the resources of each RML module.

3.1. RML-Core

RML-Core contains the core mechanism of transforming data into an RDF Knowledge Graph, such as RML Triples Maps and Predicate Object Maps. These core mechanisms were inherited mainly from R2RML [4] and the original RML specification [3]. When implementing BURP according to the RML-Core specification, we discovered several problems with the test cases, such as incompatibilities with other standards and inconsistencies between test cases. We opened an issue, discussed it with the community on GitHub to establish a solution, and applied the solution in the RML-Core repository for each problem.[2]

Compatibility with other (Web) standards was lacking among test cases. RML test cases using JSONPath as their reference formulation to refer to data from the RML mappings did not follow the latest IETF JSONPath specification.[3] When the original RML specification was introduced, a standardized JSONPath specification did not exist. Thus, many libraries implemented JSONPath slightly differently in terms of syntax and supported features. The development of BURP and CARML[4], another RML implementation, both strived to support the new JSONPath specification and thus uncovered this incompatibility. This problem was solved in the GitHub repository of RML-Core by updating the test cases.

Ill-formed language tags must be removed by an RML Processor when generating RDF according to the RML-Core specification since each language must be well-formed for RDF Literals according to BCP47.[5] Test cases were provided to validate that Processors must enforce well-formed language tags, thus removing ill-formed language tags. However, the ill-formed examples in the test cases were, in fact, well-formed. Therefore, the test case did not achieve its goal of validating the compliance of RML Processors with the specification regarding ill-formed language tags. We discovered this problem as BURP did not report an ill-formed language tag when processing these test cases, leading us to investigate the problem.

Error handling by test cases was not clearly described. Test cases provided empty output files to test whether an RML mapping should cause a Processor to fail, but an empty file may very well be a valid output (an empty graph). Some test cases provided "partial" output to assess an RML Processor's capability to continue generating RDF in case of an error. Together with the community, we decided that RML Processors must return a non-zero exit code when errors are encountered, which was not considered before. Before this, existing RML Processors would not cause scripts to report errors to developers as they always returned a zero exit code, even if the RML mapping failed. BURP always reports a non-zero exit code in case of failure and a zero exit code on success. We are considering with the community to establish a list of non-zero exit

[2] rml-core#75 rml-core#83, rml-core#89 rml-core#93, rml-core#100, rml-core#101 rml-core#109
[3] https://ietf-wg-jsonpath.github.io/draft-ietf-jsonpath-base/draft-ietf-jsonpath-base.txt
[4] https://github.com/carml/carml
[5] https://www.rfc-editor.org/rfc/rfc5646

codes for RML Processors with a predefined meaning to communicate errors, e.g., validation or access errors properly. Such an approach would allow RML implementation to retain and generate partial results.

Incomplete coverage was quickly discovered as test cases and specifications did not properly cover certain edge cases. For example, selecting attributes in XML data, data type conversions, handling of multi-valued Term Maps, and RML's Datatype Map did not have a test case. Implementing BURP as a reference implementation forced the community to reconsider their decisions to ensure all edge cases are covered in specifications, test cases, and SHACL shapes.

Inconsistencies among relational databases are assumed to be non-existent by relying on the ANSI SQL 2008 standard, inspired by R2RML [4], which only provided test cases for the standard without considering database vendors. While many relational databases support ANSI SQL 2008, they do not always implement the standard completely and with quirks. Moreover, relational databases such as MySQL require a special configuration to enable ANSI SQL. In the past, the RML test cases assumed that the RML Processor must handle these quirks. If not, the RML Processor was considered non-compliant with the RML-Core specification. However, this is against the idea of test cases that only verify whether an RML Processor can retrieve data from a relational database and transform it into RDF. We are now more conservative; we remove and adapt test cases to handle these quirks correctly and are considering moving source-specific test cases to RML-IO. A prime example of that is MySQL trimming white spaces for NCHARs.

Base IRI configuration of the RDF output generated from an RML mapping is impossible. An RML and R2RML mapping may contain a base IRI for the mapping, which was wrongly assumed to be the same base IRI for the RDF output generated by the RML mapping. This assumption was even stronger through the RML test cases since the R2RML test cases[6] inspired them. Unfortunately, the R2RML test cases used the same base IRI as the base for the generated RDF (to be provided as input to the R2RML processor according to the R2RML specification) and the base of the mapping. There is currently an ongoing discussion[7] within the W3C Community Group to allow the specification of a base IRI independent of the base IRI for the mapping.

SHACL shapes validation of RML mappings contained small mistakes such as a misleading description, wrong predicates, or wrong node kinds. Thanks to the community, we can solve these SHACL shape problems to reduce the burden on other developers of RML processors.

3.2. RML-IO

RML-IO handles all access with data sources by introducing an RML Logical Target [5] and separates the RML Logical Source from the RML-Core in the original RML specification [3]. RML IO provides data access descriptions and aligns them with RML to allow data integration from various heterogeneous data sources into RDF. Only a single RML Processor (RMLMapper [3]) fully complies with the original RML specification regarding Logical Source and Logical Target. Therefore, some problems were not discovered in the past[8] in the test cases, such as small typos in the test cases, relative file paths, missing data type inference for other heterogeneous data sources, etc.

[6] https://www.w3.org/2001/sw/rdb2rdf/test-cases/

[7] rml-core#30

[8] rml-io#36 rml-io#46, rml-io#47, rml-io#51, rml-io#53, rml-io#57, rml-io#58

Relative file paths were not considered when deprecating string Literals in `rml:source` by [6]. Relative file paths are not possible in access descriptions that use IRIs as file paths, such as DCAT [7], because they require a base IRI for the specific Source or Target to access. This base IRI is then used to resolve the relative IRI into an absolute IRI according to RFC 3986.[9] Although this deprecation had already been in effect for a decade [6], it was not performed to maintain backward compatibility with existing RML mappings with the original RML specification. Thus, the problem of relative file paths did not emerge until the original RML specification was refactored into modules at the W3C Community Group. During the refactoring, the deprecated string Literals in `rml:source` option was removed from the RML-IO specification, and access descriptions, e.g., DCAT [7], SD [8], and CSVW [9], were added. One way to solve this was adding a base IRI to every RML Logical Source and Logical Target. To avoid confusion with the base IRI of the RDF output and the RML mapping, we introduced a new class, `rml:RelativePathSource`, which allows describing a relative path using `rml:path` and `rml:root` to specify the root folder of the relative path. If the root path is not provided, it defaults to the current working directory of the Processor. BURP supports this new class `rml:RelativePathSource`, as most test cases heavily use this to specify the input data for each test case. Hopefully, this new class can support all edge cases regarding relative paths.

Datatype inference was defined by the R2RML and the original RML specification to extract datatypes from relational databases by [R2]RML Processors. Still, such approaches for other heterogeneous data sources were not specified in RML. RML Processor developers proposed different approaches to determine a value's datatype. This caused RML Processors to generate different RDF using the same RML mapping, such as a '64bit integer' compared to a '32bit integer' when the original data source specified an 'integer' as a datatype. Datatype inference is still under discussion[10] and is considered to be moved to RML-IO to have all different data sources in RML-IO instead of RML-Core since RML-Core test cases should only focus only on the core mechanisms of RML instead of data source access descriptions. This method reduces the number of test cases in RML-Core as they are replicated for every data source. This problem was discovered as BURP does not assume anything and strictly follows the provided specifications. The community is considering specifying the datatype inference and corresponding reference formulations in a document to avoid inconsistencies among RML Processors when interpreting reference formulations and performing datatype inference.[11]

Under-specified vocabulary Since BURP only implements what is defined by the RML specifications, we noticed that the RML-IO vocabulary was under-specified. For example, the interpretation of `rml:encoding` with compressed files can be interpreted in 2 ways[12]: (i) is it the encoding of the compressed file, (ii) or the encoding of the file being compressed? Therefore, the vocabulary must be more specific to define which file the encoding applies to.

Compatibility with other (Web) standards was also problematic for RML-IO when JSON-Path expressions were used. The test cases did not follow the latest IETF JSONPath specification, similar to RML-Core (Section 3.1).

[9] https://www.rfc-editor.org/rfc/rfc3986#section-4.1
[10] rml-io#87
[11] rml-core#113
[12] `rml:encoding` is only applicable if the original data access description does not define the encoding.

3.3. RML-CC

RML-CC is the RML module that supports generating RDF Collections (`rdf:List`) and Containers (`rdf:Alt`, `rdf:Bag`, and `rdf:Seq`). This module is completely new, and no Processor exists at the time of writing to support the RML-CC module. However, this module drew inspiration from two *predecessors*: (i) an extension of R2RML-F [10] for RDFS Collections and Containers for R2RML [11], and (ii) xR2RML [12], which is another extension of R2RML that supported non-relational data. The first proposed an approach that supported collecting terms across iterations and of different term types. The latter provided more control and allowed the generation of Collections and Containers from various sources. Both approaches were consolidated, and further functionality was provided for generating Collections and Containers from a Subject Map, empty Collections and Containers, etc., formulated as a set of requirements in [1]. No prior test cases existed for RML-CC, thus they were all created when the RML-CC specification was written. However, when implementing RML-CC in BURP, we still encountered some problems[13], e.g., incomplete coverage or RML mapping validation errors from the SHACL shapes.

Incomplete coverage of test cases as some edge cases only emerged implementing the RML-CC specification in BURP. We discovered that the test cases did not include gathering the terms generated by a Referencing Object Map. We also realized that the test cases assumed different base IRIs to generate RDF compared to other RML modules. While this is not an issue, the lack of coherence renders the suite of test cases less comprehensive to developers implementing an RML Processor.

SHACL validation errors with RML mappings were introduced by small changes in the vocabulary as contributors changed the specification and the corresponding test cases of other RML modules but did not update the SHACL shapes. Therefore, BURP raised mapping validation errors on some RML-CC test cases. Problems in SHACL shapes were resolved in their respective repositories.

Compatibility with other (Web) standards, similar to RML-Core (Section 3.1) and RML-IO (Section 3.2), updating to the latest IETF JSONPath specification was only discovered after implementing RML-CC in BURP.

3.4. RML-Star

The RML-Star module [13] was developed to support generating RDF-Star [14] around the same time the new RML modules were proposed. It was initially conceived for the original RML specification and later adapted to the new RML modules. RML-Star is currently only supported by a single RML Processor (Morph-KGC^{star} [15]). RML-Star had supporting test cases that had already been verified during the implementation of Morph-KGC. However, we discovered some problems when we started working on BURP.

Relative paths String Literals were deprecated and removed in RML-IO (cf. Section 3.2). However, the existing RML-Star test cases were not adjusted, and string literals were still used for the relative file paths of the test cases' input data. We solved this problem for all RML-Star

[13] rml-cc#43, rml-cc#46, rml-cc#47

test cases in their GitHub. repository[14]

Validating RDF-Star output is currently challenging as not all existing RDF libraries, such as RDFLib[15], support RDF-Star. This situation is improving as the RDF community implements RDF-Star in existing RDF libraries.

3.5. RML-FNML

RML-FNML module aligns RML with FnO data transformations [16] to allow data transformations to be performed on the data during the mapping into RDF. FNML was heavily revised in the new RML modules to handle edge cases, e.g., multiple reference values, arrays, and other use cases. Morph-KGC [15] recently implemented support for the new RML-FNML module and verified its compliance through the corresponding RML-FNML test cases. However, when implementing BURP, and by only focussing on the specification, several problems[16] emerged:

The deprecation of **relative paths** as string Literals (Section 3.2) also affected the RML-FNML test cases. We adapted the existing test cases[17] to use the `RelativePathSource` class instead of string Literals for `rml:source`.

Inconsistencies between SHACL shapes, the specification, and test cases, such as incorrect SHACL shapes to validate the test case's RML mapping, small mistakes in predicate names, or specifying the behavior of multi-value edge cases. These small problems were discovered during BURP's development, reported to and resolved by the RML-FNML maintainers.

4. Conclusion

Through this work, we increased the quality, coverage, and correctness of the RML test cases, specifications, and SHACL shapes by implementing a reference implementation 'BURP' (Basic and Unassuming RML Processor) of the new RML modules and checking the BURP's compliance with the RML modules. This way, we discovered several problems, e.g., mismatches between specifications, test cases, vocabulary, and edge cases that were not covered.

Thanks to this work, developers of RML Processors will encounter fewer problems complying with the new RML specifications and achieve a more consistent execution behavior. This exercise has also informed us and the community about the need for further extension and introduction of test cases, depending on consensus within the community. In the future, we aim to extend BURP further together with the community to cover all RML modules, demonstrating the community's integral role in the ongoing development process.

Acknowledgments

Dylan Van Assche is supported by the Special Research Fund of Ghent University[18] under grant BOF20/DOC/132. The collaboration of Dylan Van Assche and Christophe Debruyne is

[14] rml-star#27
[15] https://github.com/RDFLib/RDFlib
[16] rml-fnml#33, rml-fnml#34
[17] rml-fnml#35
[18] https://www.ugent.be/en/research/funding/bof/overview.htm

stimulated by the KG4DI FWO scientific research network (W001222N).

References

[1] 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: The Semantic Web - ISWC 2023 - 22nd International Semantic Web Conference, Athens, Greece, November 6-10, 2023, Proceedings, Part II, volume 14266 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 152–175.

[2] 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).

[3] 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: Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014, volume 1184 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014.

[4] 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/.

[5] D. Van Assche, G. Haesendonck, G. De Mulder, T. Delva, P. Heyvaert, B. De Meester, A. Dimou, Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation, in: M. Brambilla, R. Chbeir, F. Frasincar, I. Manolescu (Eds.), Web Engineering, 21st International Conference, ICWE 2021, Proceedings, volume 12706 of *Lecture Notes in Computer Science*, Springer, Cham, 2021, pp. 337–352.

[6] 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.

[7] F. Maali, J. Erickson, Data Catalog Vocabulary (DCAT), Recommendation, World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/vocab-dcat/.

[8] W. Gregory Todd, R. P. Institute, SPARQL 1.1 Service Description, Recommendation, World Wide Web Consortium (W3C), 2013. URL: https://www.w3.org/TR/sparql11-service-description/.

[9] P. Rufus, T. Jeni, K. Gregg, H. Ivan, Metadata Vocabulary for Tabular Data, Recommendation, World Wide Web Consortium (W3C), 2015. URL: https://www.w3.org/TR/tabular-metadata/.

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

[11] C. Debruyne, L. McKenna, D. O'Sullivan, Extending R2RML with support for RDF collections and containers to generate MADS-RDF datasets, in: Research and Advanced Technology for Digital Libraries - 21st International Conference on Theory and Practice of Digital Libraries, TPDL 2017, Thessaloniki, Greece, September 18-21, 2017, Proceedings, volume 10450 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 531–536.

[12] F. Michel, L. Djimenou, C. Faron-Zucker, J. Montagnat, Translation of Relational and Non-relational Databases into RDF with xR2RML, in: WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015, SciTePress, 2015, pp. 443–454.

[13] J. Arenas-Guerrero, A. Iglesias-Molina, D. Chaves-Fraga, D. Garijo, O. Corcho, A. Dimou, Declarative generation of RDF-star graphs from heterogeneous data, Submitted to Semantic Web (2024).

[14] H. Olaf, C. Pierre-Antoine, K. Gregg, S. Andy, RDF-star and SPARQL-star, Final Community Group Report, World Wide Web Consortium (W3C), 2021. URL: https://w3c.github.io/rdf-star/cg-spec.

[15] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, Semantic Web 15 (2024) 1–20.

[16] B. De Meester, T. Seymoens, A. Dimou, R. Verborgh, Implementation-independent Function Reuse, Future Generation Computer Systems 110 (2020) 946–959.