The Conformance of an RML Processor Built from Scratch to Validate Specifications and Test Cases

Christophe Debruyne²*,†, Dylan Van Assche¹*,†

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

Abstract
The Knowledge Graph Construction community has worked on the new RML specifications for the past few years, consolidating and refining various [R2]RML extensions to support various use cases. This new specification involved scholars and practitioners in one or more of RML’s modules. These modules were independently specified (vocabulary, SHACL shapes, and test cases). Moreover, participants in the Knowledge Graph Construction Workshop Challenge usually adapted their existing implementations, which have been developed (often to support research projects) to support specific problems (e.g., rewriting mappings and distributed processing). Rather than starting from existing implementations, which come with an inherent bias, we propose developing an RML Processor from scratch to avoid this bias. This engine aims to support the new RML specification while not being influenced by the prior [R2]RML implementations. We report on implementing the Basic and Unassuming RML Processor (BURP) and the current state of RML compliance. While the impact of BURP has been reported in more detail elsewhere, we hope that BURP will become the reference implementation for other implementations.

Keywords
RML, RML Conformance Checking, Knowledge Graph Generation, BURP

1. Introduction
Over the past few years, the Knowledge Graph Construction community has dedicated significant effort to consolidate and refine various [R2]RML extensions into a new, comprehensive RML specification [1]. This endeavor involved collaboration between scholars and practitioners across multiple RML modules (vocabulary, SHACL shapes, and test cases) and aimed to address a broader range of use cases. Notably, participants often adapted existing implementations developed for specific research projects to address unique challenges (e.g., optimizing the RDF generation process by rewriting mappings and distributed processing of RDF generation). Tools often rewrite or extend implementation and aim to ensure some form of backward compatibility. RMLMapper¹, for instance, now supports R2RML [2], the original RML [3], its various

---

¹https://github.com/RMLio/rmlmapper-java
extensions, and now the new RML-core specification. One could argue that this complicates things.

In contrast to this approach, we propose the development of an RML Processor from scratch, unburdened by the inherent biases\(^2\) of prior [R2]RML implementations. The Basic and Unassuming RML Processor (BURP) is designed to support the new RML specification. While the impact of BURP has been documented elsewhere \([4]\), we anticipate that it will serve as a baseline for future RML implementations. This paper reports on the implementation of BURP and assesses its current compliance with the RML specification, which was covered by Track 1 of the KGC Workshop's Challenge.\(^3\)

### 2. Implementation

The development of BURP was initially driven by the lack of an RML-CC implementation, which consolidated and expanded ideas presented in \([5]\) and \([6]\). Implementing RML-CC, which allows for values to be aggregated into RDF Containers and Collections within and across iterations, required drastic changes in existing codebases of RML implementations. Moreover, we discovered inconsistencies within and across RML modules. Thus, an implementation started from scratch seemed an adequate approach to developing this RML Processor to avoid these inconsistencies and evaluate the community’s decisions in the new RML modules.

"Basic" refers to the implementation following steps that are inspired by R2RML’s reference algorithm: it uses simple data structures, relies on nested loops, does not rewrite mappings, does not use concurrent or distributed programming techniques, etc. BURP assumes that all data to be transformed fits in the machine’s memory. The algorithm is deliberately kept simple, as introducing the aforementioned techniques may come at a cost. For example, the aggregation of results in a distributed environment relies on associative monoids, which affect the results yielded by Gather Maps. "Unassuming" means that the specification, the shapes, and the test cases solely drive the development of this processor and no other assumptions w.r.t. the mappings and data are made to optimize the RDF generation process.

BURP follows simple steps to generate RDF, similar to the R2RML reference algorithm. The code and RDF generation algorithm is deliberately kept simple to help RML Processors’ developers implement the new RML specifications. BURP uses simple data structures to store all data in memory. Moreover, BURP does not try to recover from or correct errors. When an error occurs, BURP merely outputs an error message to the user indicating what went wrong and, if possible, where it went wrong. Then, it exits with a non-zero exit code. BURP will not even try to generate partial results. Such a feature is potentially and arguably desirable in industry settings as one only needs to rerun the failed mappings. Still, we deem this beyond the scope of a reference implementation.

---

\(^2\)I.e., relying on codebases developed for (or starting from) different mapping languages, [R2]RML dialects, and/or developed for specific use cases.

\(^3\)https://kg-construct.github.io/workshop/2024/challenge.html
3. Compliance

BURP currently fully supports RML-Core, RML-CC, and RML-FNML. It also supports some functionality of RML-IO\(^4\). Complete support for RML-IO and RML-Star is planned for the latter part of 2024.

BURP passes 100% of the RML-Core and RML-CC features and 92% of RML-FNML. The only failed test is RMLFNOTC0000-CSV6. This test relies on generating a UUID via a function that takes no input. The generation of the same UUID is practically unlikely, and the test thus always fails. Given that this test was ill-conceived, we deem that we cover the RML-FNML specification regarding the test cases.

We noticed issues when running RML-IO test cases; some tests relied on outdated RML, and others contained mistakes. RMLSTC0006c, for instance, relied on an endpoint that was not configured properly. As such, BURP allowed us to determine what went wrong. BURP passes 78% percent of RML-IO source test cases, with the ones failing related to different vocabularies to download data and CSVW dialects. Only one of the RML-IO target test cases passes (2%).

4. Discussion: Are We Sufficiently Conservative?

It is important to remember that the specifications and test cases mostly drive the development of BURP. This has led to the various RML modules becoming more integrated (e.g., the SHACL shapes must consider the various RML Term Maps and Expression Maps).

Several questions can be raised concerning the coverage of our test cases. Some examples that can be mentioned are: 1) How can we ensure that we have covered most (if not all) combinations of the various modules? 2) RML-FNO uses rml:inputValueMap to link an input with a Term Map. Some Term Maps have a Graph Map (e.g., Subject Maps), how does that impact the Predicate Object Maps with Graph Maps? 3) Quoted triples can be included in RDF Containers and Collections, but what is the expected behavior when RML Quoted Triples Maps are also used as a Gather Map?

We recognize that some of these corner cases seem far-fetched, but they require documentation. The behavior can be left to the implementation, but our opinion is to propose documenting the expected behavior via notes and test cases.

5. Conclusions

We reported on the implementation of a simple RML Processor ‘BURP’ that was initially driven by a need for an RML-CC-compliant processor and turned out to be an exercise to test the validity of various RML modules throughout the Knowledge Graph Construction Workshop Challenge.

This RML Processor, which we dubbed BURP, is not (necessarily) intended as a production-ready tool as everything is processed in memory. No effort was spent on optimizing the RDF generation process (no mapping rewriting, no parallel processing, etc.). We hope that BURP will become the community’s reference implementation and sandbox for further research.

\(^4\)RML Logical Sources are supported, and RML Logical Target is under development
Acknowledgments

Dylan Van Assche is supported by the Special Research Fund of Ghent University\(^5\) under grant BOF20/DOC/132. The collaboration of Dylan Van Assche and Christophe Debruyne is stimulated by the KG4DI FWO scientific research network (W001222N).

References


\(^{5}\)https://www.ugent.be/en/research/funding/bof/overview.htm