Linking Meaning: Semantic Technologies Shaping the Future of AI
B. Spahiu et al. (Eds.)

doi:10.3233/SSW250007

Declarative Generation of RDF Collections and Containers from Heterogeneous Data

Christophe DEBRUYNE [a], Souail JAADARI [a] David CHAVES-FRAGA [b,c]
[a] *Montefiore Institute, University of Liège, Belgium*
[b] *Intelligent Systems Group, Universidade de Santiago de Compostela, Spain*
[c] *CiTIUS, Universidade de Santiago de Compostela, Spain*

Abstract.
Purpose: This paper addresses the lack of practical support for generating RDF Collections and Containers from heterogeneous sources in existing mapping tools with the RDF Mapping Language (RML). While the RML Collections and Container (CC) module defines their generation, RML-CC implementations remain limited to a reference implementation called BURP that was not conceived for efficiency or scalability. We aim to close this gap by extending a tool for efficient RML generation with support for RML-CC.
Methodology: We extended Morph-KGC to support RML-CC and developed YARRRML-CC for user-friendly mapping definitions. We also updated Yatter to enable translation from YARRRML-CC to RML-CC. We validated these tools using 35 RML-CC test cases and 22 additional YARRRML-CC cases and conducted performance evaluations using synthetic datasets.
Findings: While BURP passes all RML-CC test cases and scales up to 1M records, Morph-KGC passes only 51% due to architectural constraints and struggles with larger datasets—Morph-KGC's reliance on Pandas limits support for complex constructs such as nested collections. Yatter fully supports YARRRML-CC. Morph-KGC performs well in standard RDF generation but struggles with RML-CC at scale. This highlights the importance of selecting tools that align with the structural complexity and performance demands of specific use cases.
Value: Our work enhances the practical applicability of RML-CC in knowledge graph construction by providing different tooling supports (BURP for the Java ecosystem, and Morph-KGC for complete Python pipelines), interoperability through YARRRML-CC, and validated performance insights.

Keywords. RML, KG Construction, Collections and Containers

1. Introduction

RDF (Resource Description Framework) offers native constructs, namely Collections and Containers, to represent ordered and grouped data[1]. These structures are crucial in many knowledge graph scenarios, where the preservation of order or grouping semantics

[1] https://www.w3.org/TR/rdf-schema/#ch_collectionvocab

is essential (e.g., lists of authors, sequences of events, or sets of related measurements). Despite their relevance, current RDF generation practices rarely include Collections and Containers due to the limited support in existing declarative mapping languages [1,2] and engines.

While declarative mapping languages such as R2RML [3] and RML [4] have become standard approaches for generating RDF from structured and semi-structured data, they lack native support for RDF Collection and Container constructs. As a result, practitioners are often forced to resort to ad-hoc scripting or post-processing steps, which reduces the transparency, maintainability, and interoperability of the data generation process. Moreover, mainstream mapping engines, including FlexRML [5], SDM-RDFizer [6], and Morph-KGC [7], have traditionally focused on "flat" graph generation, with no built-in mechanisms to capture order or grouping semantics beyond basic subject-predicate-object relationships.

This paper addresses this gap by presenting a complete ecosystem for the declarative generation of RDF Collections and Containers from heterogeneous data sources. We extended[2] Morph-KGC [7] to support the RML-CC module [4]. Our extension will be compared against BURP [8], which was the only implementation supporting RML-CC. We also extend the human-friendly serialization, YARRRML [9], to include YARRRML-CC, enriching it with constructs for defining RDF lists, bags, sequences, and alternatives. These extensions allow users to define collection structures explicitly and portably across mapping engines. In addition, we also extend Yatter [10], a translator that converts YARRRML into easy-to-read (R2)RML for supporting the conversion of Collections and Containers, facilitating interoperability and tooling integration.

To validate the proposed approach, we rely on the test-cases[3] defined by the W3C Knowledge Graph Construction Community Group[4] for Collections and Containers. We also adapt them into YARRRML-CC for the evaluation of Yatter's correctness and fidelity. Furthermore, we scale selected test cases to assess the performance impact of generating structured RDF constructs across multiple engines. This thorough validation demonstrates both the functional soundness and the scalability of our ecosystem, establishing a foundation for robust and declarative handling of RDF Collections and Containers in knowledge graph construction workflows.

The paper is structured as follows: Section 2 introduces the background concepts relevant to our work. Section 3 outlines the main contributions of the paper. Section 4 details the validation process and the data generation methodology. Section 5 discusses related work, and Section 6 concludes the paper and outlines directions for future research.

2. Background

In this section, we introduce the RML-CC module, which builds upon the RML mapping language [4] to enable the generation of RDF collections and containers. We also present the two KG Construction engines: Morph-KGC [7], a heuristic-based and efficient RML-compliant system, and BURP [8], a reference implementation for the new

[2] https://github.com/chrdebru/morph-kgc
[3] http://w3id.org/rml/cc/test-cases
[4] https://www.w3.org/community/kg-construct/

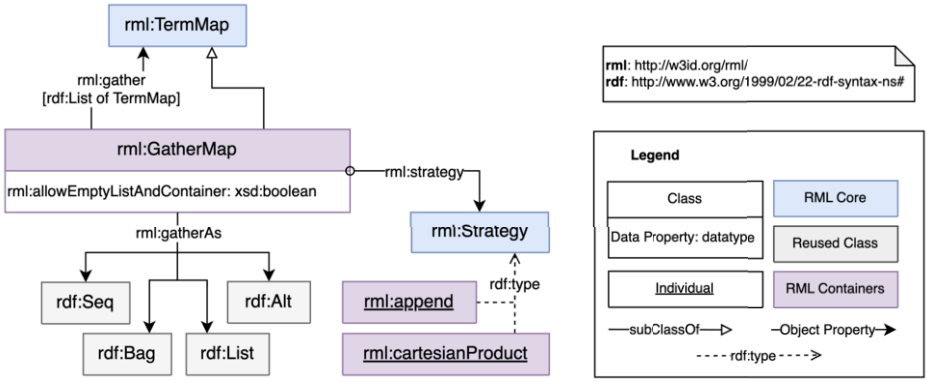

Figure 1. The RML-CC module (represented using the Chowlk Visual Notation [13]). The RML-CC resources are highlighted in purple, while the rest of the represented ontology belongs to the RML-Core module.

RML specification [4] which has been written from scratch to have no influence from prior implementations of RML. These tools provide complementary approaches to processing RML-CC mappings and allow us to compare their behavior under varying data configurations and mapping constructs.

2.1. RML Collections and Containers

RML-CC is the RML module that supports generating RDF collections and containers. This module drew inspiration from two (R2)RML-like predecessors: (i) an extension of R2RML-F for RDFS Collections and Containers [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 into RML-CC, 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 [4]:

1. Collect values from one or more (multi-valued) Term Maps.
2. Control the generation of empty collections and containers.
3. Generate nested collections and containers.
4. Group different term types.
5. Generate collections or containers as a subject.
6. Assign an IRI or blank node identifier to a collection or container.

An overview of RML-CC is presented in Figure 1, along with a representative example in Listing 1. The module introduces one main class, `rml:GatherMap`, which defines three properties (`rml:gather`, `rml:gatherAs`, and `rml:strategy`) used to express the requirements described above. RML-CC proposes two strategies to process gathering values from multiple multi-valued term-maps: `rml:append`, the default strategy, and `rml:cartesianProduct`. While `rml:append` is arguably the most intuitive approach, W3C Community Group proposed the use of strategies so that engines can

Listing 1: Example of RML-CC mapping document in Turtle syntax

```
@prefix rml: <http://w3id.org/rml/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ex:  <http://example.com/ns#>.
@base        <http://example.com/>.

<#TM> a rml:TriplesMap;

  rml:logicalSource [
    rml:source [ a rml:RelativePathSource ; rml:path "data.json" ] ;
    rml:referenceFormulation rml:JSONPath ;
    rml:iterator "$.*" ] ;

  rml:subjectMap [ rml:template "e/{$.id}" ] ;

  rml:predicateObjectMap [
    rml:predicate ex:with ;
    rml:objectMap [
      rml:gather ([ rml:reference "$.values.*" ]); rml:gatherAs rdf:Bag ]] .
```

suggest and implement bespoke strategies, e.g., "append without duplicates". Although gathering terms across iterations is not explicitly stated as a requirement, it is implicitly supported through the use of blank nodes or IRIs to represent collections or containers, in combination with the `rml:append` strategy. When an RML engine constructs such structures, the terms yielded in each iteration are appended accordingly. For further details on this module, we refer the reader to the seminal paper on RML [4].

2.2. *Morph-KGC and BURP*

Morph-KGC [7] is an open-source (R2)RML engine used for constructing RDF Knowledge Graphs from structured and semi-structured data. Morph-KGC is implemented in Python and uses Pandas data frames for data manipulation and processing. It is designed with a modular architecture to handle the different stages of knowledge graph construction. Morph-KGC's mapping partitioner, for instance, analyses the mappings to identify independent groups of rules that can be executed in isolation. This optimization reduces memory consumption and allows for parallel processing. Pandas offers functionality that affects efficiency, e.g., the efficient removal of duplicates. Its reliance on Pandas at times comes at a cost. For instance, when a reference object map relies on multiple join conditions, Morph-KGC performs less well when processing "simple" joins, as Pandas relies on a separate algorithm to perform that join [14].

BURP [8] is a Java-based application whose development has been closely tied to the RML test cases[5]. It was used to test and validate the new RML specification and its test cases [4], helping identify and address inconsistencies or issues within the specification. BURP is the W3C Community Group's effort to develop an RML reference implementation. Currently, BURP is the only RML engine fully supporting RML-CC. By starting from scratch, it aimed to avoid biases in existing RML engines, which often

[5] https://w3id.org/rml/portal

evolved from research projects with specific goals (i.e., for optimization and distributed processing [6,7]). BURP uses simple data structures, storing all data in main memory. This means it is not optimized for large datasets that might exceed available RAM. The RDF generation algorithm is deliberately kept simple, similar to the R2RML reference algorithm. No complex optimizations, mapping rewrites, parallelization, or distributed computing techniques are applied. This simplicity makes it easier to implement the various aspects of RML. Finally, BURP does not attempt to recover from or correct errors. If an error occurs, it exits with a non-zero exit code, providing an error message to the user. Unlike RML engines such as RMLMapper, it does not try to produce "best-effort" partial results. It suffices to look at distributed processing to motivate such an implementation that runs in one thread (and thus machine); when mapping tasks are sent to different cluster nodes, the results reported back may not necessarily be received in their natural order. As such, the output is not guaranteed to be deterministic.[6]

3. Declarative generation of RDF Collections & Containers

RML-CC introduces explicit support for generating RDF Collections and Containers from heterogeneous data sources. While these RDF structures are commonly used, most existing mapping engines offer limited support. This section reviews how RML-CC is implemented in two different RML engines. We present BURP, designed as the reference implementation and the partial support available in Morph-KGC. We also describe the extension of YARRRML to include RML-CC features in its syntax, and how Yatter (a YARRRML-compliant engine) has been updated to translate between YARRRML-CC and standard RML-CC mappings.

3.1. RML-CC in BURP

The implementation of BURP closely follows R2RML's algorithm, which has been extended for RML. While we will not delve into the details of supporting other extensions, such as RML-FNML and RML-STAR, the support for RML-CC arguably requires the most changes to the original algorithm. The following description furthermore describes how RML-CC was implemented in BURP.

It is important to note that a Gather Map generates graphs, not terms. Take, for example, a Subject Map as sm. When sm is not a Gather Map, then all of its terms (because Term Maps may be multi-valued) are processed as usual. When a sm is a Gather Map, however, the engine generates a set of graphs containing the collections and containers. Each "sub-graph" holds a reference to the term representing the collection or container and the graph with all the triples pertaining only to that collection or container. For each sub-graph, the node is added to the list of subjects, and the graphs are added to the datasets using the list of named graphs generated by sm's Graph Map. Gather Maps can be nested. When dealing with a Gather Map, we invoke the function "generate[GatherMap]Graphs" instead of "generateTerms," which is responsible for passing

[6] In Apache Spark, for instance, for the `reduce` function to work correctly and *consistently* in a distributed environment, the function provided to `reduce` must be both associative and commutative. Spark cannot guarantee a deterministic or correct final result if it is not.

the iteration on its Term Maps (”regular” or themselves Gather Maps). The same logic is applied to the generation of collections and containers in Object Maps.

Containers have a type (Bag, Seq, or Alt), so empty containers are easy to manage as their IRIs or blank node identifiers are used within at least one triple. This is not the case for empty lists. Empty lists, which may in the end not be empty, cannot be identified as `rdf:nil`. We internally use an artificial list “type” which is removed during the RDF generation process.

BURP is implemented using Apache Jena. In Apache Jena, the members of a container are linked via `rdf:_x` properties, which thus requires some complex and arguably inefficient bookkeeping to connect elements of these containers generated via different gather maps and/or generated in various iterations.

3.2. RML-CC in Morph-KGC

To support the RML-CC specification, Morph-KGC incorporates specific logic within its materialization layer to handle `rdf:List`, `rdf:Bag`, `rdf:Seq`, and `rdf:Alt`. This includes the interpretation of the `rml:gather`, `rml:gatherAs`, and `rml:strategy` predicates, which control the aggregation, naming, and ordering of collection members. At this moment, our implementation only supports `rml:append`, which we deem the most fitting approach to support. Dedicated helper functions were introduced to construct RDF triples for both anonymous and named collections and containers, with special attention to correctly managing index assignments in sequences. These enhancements ensure syntactic and semantic compliance with the RML-CC module, despite the limitations of the broader execution strategy.

However, Morph-KGC provides partial support for RML-CC, since its internal architecture introduces notable limitations. The engine heavily relies on pandas for data processing, which is optimized for flat, tabular data structures. This makes it difficult to handle nested or hierarchical constructs such as the ones used for generating RDF Collections and Containers. In particular, features such as processing multivalued objects or removing duplicates within the same array require collecting and structuring values across multiple rows, which conflicts with the row-wise evaluation model used in Morph-KGC. As a result, while basic functionality is in place, the support remains constrained and introduces significant overhead and complexity when processing mappings involving these advanced features.

The execution planning strategy employed by Morph-KGC [7] is also not very well-suited for supporting the efficient execution of RML-CC mapping. This planning mechanism operates at the level of individual mappings and relies on simplified representations of Predicate-Object Maps (POMs), which do not capture the structural complexity introduced by RML-CC constructs. Hence, Morph-KGC optimizations need to be adapted in order to provide better support of complex RML-CC structures and ensure high performance and scalability when processing mappings that group representations.

3.3. YARRRML-CC and Yatter

To facilitate the development of the mapping rules for generating RDF collections & containers, we have also extended YARRRML [9]. YARRRML is a human-friendly serialization of RML [10] based on the YAML language. Following the same approach as

Table 1. Relationship betweeen RML-CC vocabulary to YARRRML-CC representations

RML-CC	YARRRML-CC
`rml:gather`	`gather`
`rml:gatherAs`	`gatherAs`
`rml:strategy`	`strategy`
`rml:allowEmptyListAndContainer`	`empty`
`rml:cartesianProduct`	`cartesianProduct`
`rml:append`	`append`

in the previous extension of the RML-star module (YARRRML-star [10]), we define a set of correspondences between the RML-CC vocabulary and its YARRRML-CC representation. These correspondences are summarized in Table 1. In addition, the values of the `subject` and `object` keys must be extended to support all possible combinations introduced by the RML-CC module. An example is provided in Listing 2, illustrating the added complexity introduced by RML-CC in YARRRML-CC when expressing the generation of recursively nested structures.

Support for this extension has been integrated into Yatter [10], a YARRRML-compliant translator, starting from version 2.0[7]. The tool has been updated to parse and validate YARRRML-CC constructs, including support for nested structures and the associated keys defined in Table 1. In addition, Yatter implements now a normalization process over YARRRML mappings, which flattens syntactic sugar and expands shorthand constructs into their fully qualified equivalents. This normalization ensures a consistent and unambiguous transformation into RML. This feature is particularly important, as Yatter produces the resulting RML mapping in Turtle syntax. This serialization uses predicate object lists within blank node properties[8], as recommended by the [R2]RML specifications. Specifically designed to be human-readable, this output facilitates debugging and inspection by knowledge engineers, providing a clear view of how high-level YARRRML-CC constructs are resolved into RML-level mappings.

4. Validation

We will validate our implementation of RML-CC in two ways. First, we will assess our implementation's compliance with RML-CC via the test cases. This will give us an indication of the specification coverage. Next, we will benchmark our implementation against a naive implementation of RML-CC.

4.1. RML-CC Test Cases

Since BURP was developed from scratch with the goal of serving as a reference implementation for RML, it already supports all the test cases defined by the RML-CC module. In contrast, Morph-KGC passes 18 out of the 35 test cases (51%).[9] While this may seem underwhelming, Morph-KGC does support most of the requirements put forward

[7] https://doi.org/10.5281/zenodo.14627842

[8] https://www.w3.org/TR/turtle/#unlabeled-bnodes

[9] Some test cases are "repeated" for collections and containers, e.g., RMLTC-CC-0006-*X*. There would be 56% coverage if we accounted for that.

RMLTC-CC ID	Test	BURP	Morph-KGC	Comment
0001-Alt	Generate a rdf:Alt as an object	✓	✓	
0001-Bag	Generate a rdf:Bag as an object	✓	✓	
0001-List	Generate a rdf:List as an object	✓	✓	
0001-Seq	Generate a rdf:Seq as an object	✓	✓	
0002-Bag	Generate a named rdf:Bag as an object	✓	✓	
0002-List	Generate a named rdf:List as an object	✓	✓	
0003-EB	Allow the generation of empty bags	✓	✓	
0003-EL	Allow the generation of empty lists	✓	✓	
0003-EL-BN	Allow the generation of empty lists identified by blank node identifiers.	✓	✗	Likely a bug
0003-EL-Named	Allow the generation of empty lists identified by IRIs.	✓	✓	
0003-NEB	Empty bags are not generated	✓	✓	
0003-NEL	Empty lists are not generated	✓	✓	
0003-NELb	Empty lists are not generated II	✓	✓	
0004-SM1	GatherMap in subject map (list)	✓	✓	
0004-SM2	GatherMap in subject map (list)	✓	✗	Requires deep refactoring
0004-SM3	GatherMap in subject map (list)	✓	✗	Requires deep refactoring
0004-SM4	GatherMap in subject map (bag)	✓	✓	
0004-SM5	GatherMap in subject map (bag)	✓	✗	Requires deep refactoring
0005-App1	Append values of multiple term maps	✓	✓	
0005-App2	Append values of multiple term maps	✓	✓	
0005-Car1	Cartesian product of values from multiple term maps of a gather map.	✓	✗	Not supported
0005-Car2	Cartesian product of values from multiple term maps of a gather map.	✓	✗	Not supported
0006-IT0	Gather values from 1 term map into a collection across iterations (base case, no aggregation).	✓	✓	
0006-IT1	Gather values from 1 term map into a collection across iterations.	✓	✗	Requires deep refactoring
0006-IT2	Gather values from 2 term maps into a collection across iterations.	✓	✗	Requires deep refactoring
0006-IT3	Gather values from 1 term map into a container across iterations (base case, no aggregation).	✓	✓	
0006-IT4	Gather values from 1 term map into a container across iterations.	✓	✗	Requires deep refactoring
0006-IT5	Gather values from 2 term maps into a container across iterations.	✓	✗	Requires deep refactoring
0007-NES	Nested gather maps	✓	✗	Requires deep refactoring
0008-ROMa	Elements via reference object-map.	✓	✗	Requires deep refactoring
0008-ROMb	Elements via reference object-map with default join condition	✓	✗	Requires deep refactoring
0009-DUP-Bag	Gather duplicate values in an RDF Bag	✓	✗	Requires deep refactoring
0009-DUP-List	Gather duplicate values in an RDF List	✓	✗	Requires deep refactoring
0010-Lista	Combining graph maps and gather maps (graph map has a template and gather map does not)	✓	✗	Not supported
0010-Listb	Combining graph maps and gather maps (both graph map and gather map have a template)	✓	✗	Not supported

```
prefixes:
  ex: http://example.com/ns#
  base: http://example.com/

mappings:
 map1:
 sources:
  access: person.json
  formulation: jsonpath
  iterator: "$.*"
 s: e/$($.id)
 po:
  - p: ex:with
    o:
     value: c/$($.id)
     gather:
      - gather: $($.1.*)
        gatherAs: bag
      - gather: $($.2.*)
        gatherAs: bag
     gatherAs: list
```

```
@prefix rml: <http://w3id.org/rml/>.
@prefix ex: <http://example.com/ns#>.

<map> a rr:TriplesMap;
 rml:logicalSource [
  rml:source "data.json";
  rml:referenceFormulation rml:JSONPath;
  rml:iterator "$.*"; ];
 rml:subjectMap [
  rml:template "e/{$.id}";];
 rml:predicateObjectMap [
  rml:predicate ex:with;
  rml:objectMap [
   rml:template "c/{$.id}";
   rml:gather ([
    rml:gather ([ rml:reference "$.1.*";]);
    rml:gatherAs rdf:Bag; ] [
    rml:gather ([ rml:reference "$.2.*";]);
    rml:gatherAs rdf:Bag; ]);
   rml:gatherAs rdf:List; ];
 ].
```

Listing 2: Equivalent mappings in YARRRML-CC (left) and its corresponding translation in RML-CC (right)

by RML-CC. As we will discuss in this section, many of the problems are related to Morph-KGC's choice for a Pandas data frame as an internal data structure, and that any overhaul of the implementation may come at the cost of its strengths.

In 0003-EL-BN, the test case uses an `rml:template` to generate blank node identifiers. While generating blank nodes with "regular term maps" is possible, our implementation has an issue where the `rml:termType` directive is ignored. This is likely a bug.

RMLTC-CC-0004-SM*X* test cases are concerned with generating collections and containers from a subject map. The test cases work for collections (RMLTC-CC-0004-SM1) and containers (RMLTC-CC-0004-SM4) where there is at least one `rml:class`[10] attached to the subject map or at least one predicate-object map attached to the triples map. Those are not necessary when we merely want to generate containers and containers. The other test covers those cases, and the reason that RMLTC-CC-0004-SM2, RMLTC-CC-0004-SM3, and RMLTC-CC-0004-SM5 fail is that Morph-KGC requires all triple maps to have at least one predicate-object map.

RMLTC-CC-0005-Car*X* test cases fail as we have decided not to implement the Cartesian Product strategy. This strategy would have, moreover, required major refactoring. Similarly, Morph-KGC did not implement the gathering of terms across iterations. By relying on Pandas data frames, it is challenging to aggregate terms (or collections and containers of terms) across records, which may be distributed. Solutions to this problem

[10]It is important to note that `rml:class` is syntactic sugar for predicate-object maps with two constants: `rdf:type` for the predicate and the class for the object. Morph-KGC rewrites `rml:class` into predicate-object maps.

may also affect the performance of Morph-KGC as it limits its capabilities for distributed KG generation. The same arguments hold for collecting terms via gather maps (test cases RMLTC-CC-0008-ROM*X*) and nested gather maps (RMLTC-CC-0007-Nes). The latter is interesting as Morph-KGC adds a column with a lambda for each term map. When dealing with nested gather maps, the gather maps need to be traversed via deep recursion, and one needs to create such columns for each "leaf" and each "non-leaf," where the latter is responsible for the combinations across columns. This was impractical because of the Pandas data frame used as the primary data structure, and additional "bookkeeping" of column execution order would have been required.

RMLTC-CC-0009-DUP-*X* test cases aim to assess whether RML engines do not preprocess the documents. Morph-KGC removes duplicate values (e.g., **"a": [1, 2, 3, 2, 1]** as duplicate triples generated via "traditional" mappings are ignored. We thus notice that Morph-KGC removes duplicate values before passing them on to the expression maps.

Finally, combining gather maps with graph maps is impossible, as that would require additional "bookkeeping" that would not only hamper Morph-KGC's distributed computing capabilities but also require a major refactoring of the code base.

Summary. Looking at the requirements put forward by RML-CC, we can conclude the following:

1. Morph-KGC can collect values from one or more (multi-valued) Term Maps, but only from within the same iteration. The collection of terms across iterations requires major refactoring of Morph-KGC.
2. Morph-KGC can control the generation of empty collections and containers.
3. Morph-KGC cannot generate nested collections and containers. This would require a significant refactoring of Morph-KGC.
4. Morph-KGC can group different term types.
5. Morph-KGC can generate collections or containers as a subject. This currently requires additional POMs, but that is because Morph-KGC requires at least one POM to start the KGC generation process. This should not require a major refactoring of Morph-KGC.
6. Morph-KGC can assign an IRI to a collection or container, but there are problems when assigning blank node identifiers. This would require a significant refactoring of Morph-KGC.

4.2. YARRRML-CC Test Cases

We rely on the official RML-CC test suite proposed by the W3C Knowledge Graph Construction Community Group[11] to define a corresponding set of test cases for YARRRML-CC. These test cases aim to validate the syntactic expressiveness and translation semantics of YARRRML-CC, rather than the correctness of data generation itself. Each original RML-CC mapping was manually translated into one or several equivalent YARRRML-CC, depending on shortcuts and syntactic sugar approaches. The resulting test suite comprises 22 test cases, covering all vocabulary elements introduced by the RML-CC module, including recursive `gather` constructs, different container types via `gatherAs`,

[11] http://w3id.org/rml/cc/test-cases

Table 2. Evaluation dataset configurations by data type

Data Type	Format	Parameters	Values Used
Simple	JSON	`num_objects`	10K; 100K; 1M; 10M
		`num_lists`	Fixed at 1
		`values_per_object`	Fixed at 10
Multivalued	JSON	`num_objects`	Fixed at 2
		`num_lists`	Fixed at 2
		`values_per_list`	10K; 100K; 1M; 10M
Join	CSV	`num_rows`	Fixed at 10,000
		`join_percentage`	25%; 50%; 75%; 100%
Duplicates	JSON	`num_objects`	Fixed at 1
		`values_per_object`	Fixed at 10,000
		`duplicate_percentage`	25%; 50%; 75%; 100%

and join strategies such as `rml:append`. All test cases are publicly available and serve as a reference for evaluating conformance of YARRRML-CC-compliant tools[12]. Yatter passes all test cases successfully.

4.3. Performance Evaluation

We also aim to evaluate the performance of both engines, BURP and Morph-KGC, to understand how different parameters impact execution time [14,15]. This evaluation is based on the test cases defined above, scaling the input data sources along various parameters. All resources used in this experimental evaluation are openly available[13].

Synthetic Data Generation. To evaluate RML engines under different data conditions, we implemented a flexible data generator that produces synthetic JSON and CSV files based on configurable parameters. The goal is to create datasets that reflect common structural patterns while allowing control over volume, value multiplicity, and redundancy. We defined four types of datasets: simple, multivalued, join, and duplicates. Table 2 summarizes the configuration of each dataset type:

- Simple datasets are composed of flat JSON objects with fixed-size lists of values. The number of records scales is configurable, and in this case, we use from 10K up to 10 million, allowing us to test performance under increasing volume.
- Multivalued datasets contain two lists per object (e.g., v1 and v2). The total number of values per list is scaled in each experiment. This dataset helps assess how engines handle multivalued fields or repeated properties.
- Join datasets are constructed from a single CSV file that can be self-joined. The percentage of matching keys is configurable (in our experimentation, we use from 25% to 100%), making it possible to measure the cost of join operations.
- Duplicate datasets consist of a single object containing a large list, with a configurable percentage of duplicate values. This setup is useful for testing an engine's robustness to redundancy and how it manages repeated entries.

[12] https://github.com/citiususc/yatter/tree/main/test/rml-cc
[13] https://github.com/citiususc/rml-cc-eval

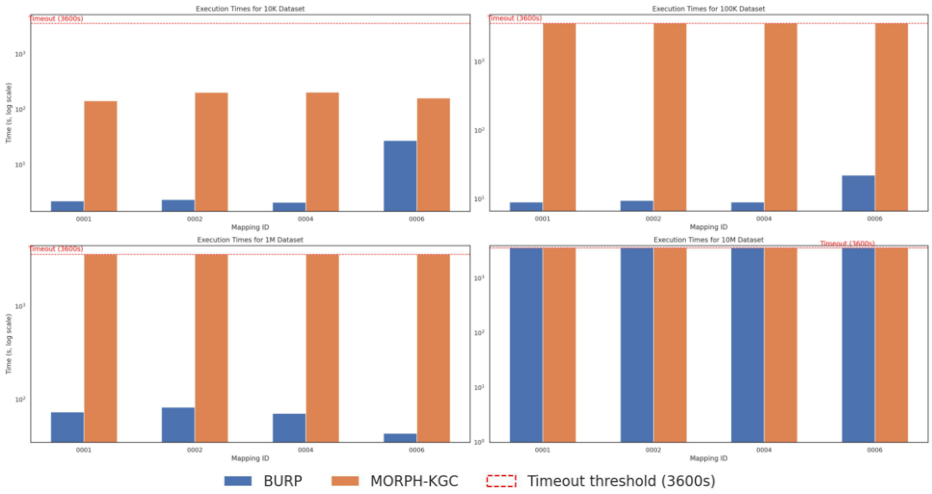

Figure 2. Execution time of simple dataset scaled in different sizes (10K, 100K, 1M y 10M).

Engines and Metrics. We use the extension implemented in Morph-KGC[14] and BURP v0.1.2[15]. For each experiment, we measure the execution time, running each configuration three times and reporting the average. The timeout is set at 60 minutes. All experiments are running on an AMD EPYC 9554 Processor with 64 cores and 755 GB RAM.

Results. We show the obtained results in Figure 2 for the simple dataset and in Figure 3 for multi-values, joins and duplicates. Figure 2 summarizes the average execution times obtained for each engine across all the simple dataset configurations. To provide a more concise and interpretable visualization, we aggregated results by grouping together mappings that share the same base identifier but differ only in their collection structure (e.g., 0001-Alt, 0001-Bag, 0001-List, and 0001-Seq are treated as a single group 0001). For each dataset size (10K, 100K, 1M, 10M), we computed the average execution time across all mappings in the group. This allows us to evaluate how each engine scales with increasing data volumes, independent of minor structural variations. As shown in the figure, BURP surprisingly demonstrates consistent and efficient performance across all dataset sizes up to 1 million objects. For the 10K and 100K object scenarios, execution times range between 2 and 10 seconds. At 1M objects, most mappings are processed in under 90 seconds, indicating acceptable scalability. However, all 10M experiments timed out, marking the current upper limit of BURP for this data type. On the other hand, Morph-KGC shows significant limitations. Most experiments either reach a timeout (exceeding the 60-minute threshold) or are simply not supported. While it manages to complete a few 10K-object cases, the execution times are disproportionately high — for instance, 160 seconds in 0001 and nearly 200 seconds in 0004, making it unsuitable for larger datasets. All 100K, 1M and 10M runs timed out.

Figure 3 presents the execution time evaluation of the BURP engine across the Multivalued, Join, and Duplicates dataset categories. The results focus solely on BURP, as

[14] https://github.com/chrdebru/morph-kgc
[15] https://github.com/kg-construct/BURP

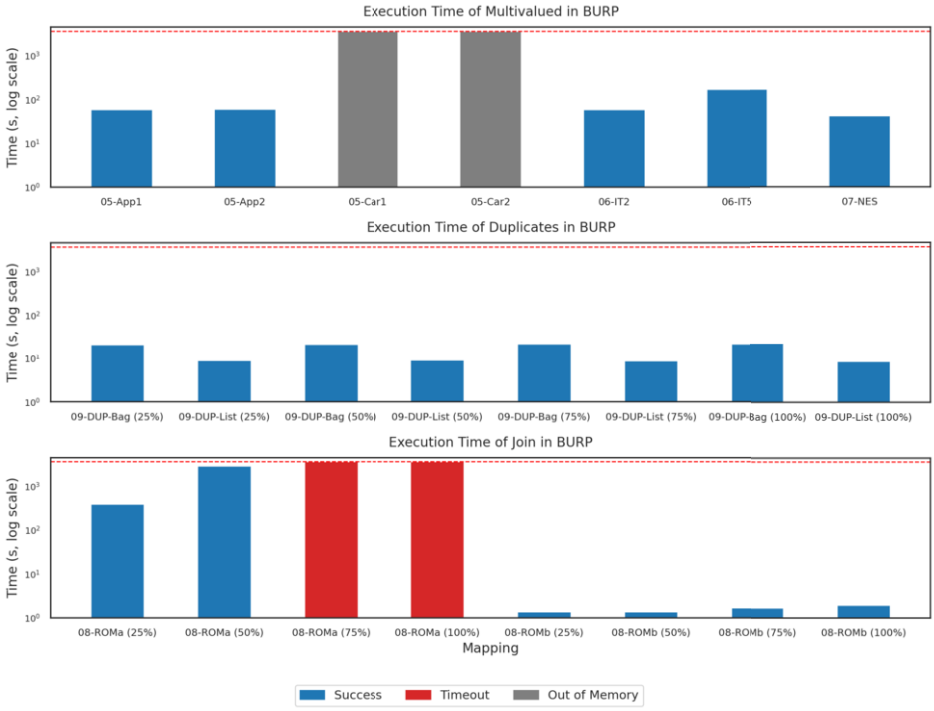

Figure 3. Execution time of multivalues, joins and duplicates dataset over BURP. In the multivalued dataset, the used size is 10K as the rest sizes (100K, 1M, y 10M) runs timed out over BURP.

the majority of these experiments are not yet supported in the current implementation of Morph-KGC for RML-CC. Unlike the simple datasets, the Multivalued experiments do not scale in BURP. All attempts to execute configurations beyond 10K values (i.e., 100K, 1M, and 10M) resulted in timeouts under BURP, indicating scalability limitations in handling multiple nested lists. For this reason, only the 10K configuration is shown in Figure 3. In the Duplicates datasets, the results show that duplication has negligible impact on BURP's execution time. This is because BURP does not include an internal deduplication mechanism (like SDM-RDFizer [6] or Morph-KGC [7]) that scales with duplication rate, processing the input uniformly regardless of redundancy. In the Join datasets, we analyze mappings with and without join conditions (e.g., ROMa vs ROMb). The execution times clearly indicate that join conditions significantly affect performance. ROMa mappings, which include one join condition, consistently require more time to process compared to their ROMb counterparts, even at the same join percentages. This highlights the overhead introduced by evaluating join relationships during materialization.

Overall, these results emphasize the limitations of current engines in scaling RML-CC mappings across more complex data patterns. The results show that BURP scales well on simple datasets up to 1M records, but fails with 10M due to memory limits. Morph-KGC only completes small cases and times out or lacks support in most experiments, highlighting its current limitations. In contrast, multivalued, duplicate, and join datasets do not scale: all attempts beyond 10K led to timeouts. Notably, duplication levels had minimal impact on performance, suggesting no deduplication mechanism in

BURP, and joins (especially with conditions, e.g., ROMa) significantly increased execution times.

5. Related Work

The recent modular RML ontology [4] introduces a formal and extensible specification for the RDF Mapping Language (RML), including the definition of optional modules such as RML-FNML, RML-CC, or RML-Star. These modules address expressiveness limitations of earlier versions, particularly in scenarios involving advanced data structures or transformation logic. Recent efforts have extended RML engines to support these advanced features required by real-world knowledge graph generation tasks. Particularly, Morph-KGC introduces support for RML-star [16], enabling the generation of RDF-star datasets from heterogeneous sources, a capability previously unexplored despite the growing adoption of RDF-star in triple stores. The system demonstrates scalability over large datasets and provides a practical implementation of the RML-star specification. In parallel, Morph-KGC has also incorporated RML-FNML [17], adding support for user-defined functions to handle complex data transformations. This enhancement allows integrating Python-based logic within mappings and improves the expressiveness and reusability of mappings in industrial contexts. These contributions significantly improve the applicability of RML-based systems to more complex and large-scale data integration scenarios.

The 2024 edition of the Knowledge Graph Construction Challenge[16] focused on evaluating the conformance of mapping engines with the latest modular RML specification through a comprehensive suite of test cases, including those targeting specific modules such as RML-CC. Several engines participated in the challenge, including SDM-RDFizer [18], FlexRML [19], Mapping Template [20], and RDFProcessingToolkit [21]. While these systems demonstrated compatibility with core RML features, only BURP successfully passed all test cases involving the RML-CC module [22]. This highlights the current lack of support for more advanced RML modules across existing engines and underscores the implementation gap that still exists in the ecosystem.

Important note to the reader. During the 6th Knowledge Graph Construction Workshop at ESWC, [23] presented a new version of SDM-RDFizer that now supports RML-CC at the 2025 edition of the Knowledge Graph Construction Challenge.[17] We were unable to compare our implementations with SDM-RDFizer due to time constraints and the fact that it was presented after the submission of our work. Nonetheless, we recognize its relevance and cite it accordingly via this paragraph.

6. Conclusions and Future Work

This paper presents a complete ecosystem for the declarative generation of RDF Collections and Containers from heterogeneous data sources. While RML-CC was defined as part of the modular RML ontology, we contribute its implementation in two engines

[16] https://w3id.org/kg-construct/workshop/2024/challenge
[17] https://w3id.org/kg-construct/workshop/2025/challenge

(BURP and Morph-KGC) and propose YARRRML-CC, an extension of the YARRRML syntax to support RML-CC features. We also update Yatter to support YARRRML-CC translation. Validation was conducted using the official W3C KGC CG test suite, which we also adapted to YARRRML-CC to verify mapping correctness and tool behavior. Additionally, we scaled the test cases to assess the performance implications of generating structured RDF constructs.

As future work, we aim to extend existing benchmarks such as KROWN [14] and GTFS-Madrid-Bench [24] to include use cases involving RDF Collections and Containers. This will enable a more thorough experimental evaluation of our approach in real-world scenarios, covering aspects like modeling expressiveness, performance, and impact on downstream tasks. These extensions will further solidify the role of structured RDF constructs in practical knowledge graph generation workflows.

Acknowledgments

The collaboration between the University of Liège and the Universidade de Santiago de Compostela is stimulated by the KG4DI FWO scientific research network (W001222N). David Chaves-Fraga is funded by the Agencia Estatal de Investigación (Spain) (PID2023-149549NB-I00), the Xunta de Galicia — Consellería de Cultura, Educación, Formación Profesional e Universidades (Centro de investigación de Galicia accreditation 2024–2027 ED431G-2023/04 and Reference Competitive Group accreditation 2022–2026, ED431C 2022/19) and the European Union (European Regional Development Fund–ERDF).

References

[1] Michel F, Djimenou L, Zucker CF, Montagnat J. Translation of relational and non-relational databases into RDF with xR2RML. In: 11th International Confenrence on Web Information Systems and Technologies (WEBIST'15); 2015. p. 443-54.

[2] Debruyne C, McKenna L, O'Sullivan D. 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 21. Springer; 2017. p. 531-6.

[3] Das S, Sundara S, Cyganiak R. R2RML: RDB to RDF Mapping Language. World Wide Web Consortium (W3C); 2012. Available from: https://www.w3.org/TR/r2rml/.

[4] Iglesias-Molina A, Van Assche D, Arenas-Guerrero J, Meester BD, Debruyne C, Jozashoori S, et al. 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. vol. 14266 of Lecture Notes in Computer Science. Springer; 2023. p. 152-75. Available from: https://doi.org/10.1007/978-3-031-47243-5_9.

[5] Freund M, Schmid S, Dorsch R, Harth A. FlexRML: A Flexible and Memory Efficient Knowledge Graph Materializer. In: European Semantic Web Conference. Springer; 2024. p. 40-56.

[6] Iglesias E, Jozashoori S, Chaves-Fraga D, Collarana D, Vidal ME. 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; 2020. p. 3039-46.

[7] Arenas-Guerrero J, Chaves-Fraga D, Toledo J, Pérez MS, Corcho Ó. Morph-KGC: Scalable Knowledge Graph Materialization with Mapping Pwartitions. Semantic Web. 2024;15(1):1-20. Available from: https://doi.org/10.3233/SW-223135.

[8] Van Assche D, Debruyne C. BURPing Through RML Test Cases. In: Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Hersonissos, Greece, May 27, 2024. vol. 3718 of CEUR Workshop Proceedings. CEUR-WS.org; 2024. .

[9] Heyvaert P, De Meester B, Dimou A, Verborgh R. Declarative rules for linked data generation at your fingertips! In: The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15. Springer; 2018. p. 213-7.

[10] Iglesias-Molina A, Chaves-Fraga D, Dasoulas I, Dimou A. Human-Friendly RDF Graph Construction: Which One Do You Chose? In: Garrigós I, Rodríguez JMM, Wimmer M, editors. Web Engineering - 23rd International Conference, ICWE 2023, Alicante, Spain, June 6-9, 2023, Proceedings. vol. 13893 of Lecture Notes in Computer Science. Springer; 2023. p. 262-77. Available from: https://doi.org/10.1007/978-3-031-34444-2_19.

[11] Debruyne C, McKenna L, O'Sullivan D. 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. vol. 10450 of Lecture Notes in Computer Science. Springer; 2017. p. 531-6. Available from: https://doi.org/10.1007/978-3-319-67008-9_42.

[12] Michel F, Djimenou L, Faron-Zucker C, Montagnat J. 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. p. 443-54. Available from: https://doi.org/10.5220/0005448304430454.

[13] Chávez-Feria S, García-Castro R, Poveda-Villalón M. Chowlk: from UML-based ontology conceptualizations to OWL. In: European Semantic Web Conference. Springer; 2022. p. 338-52.

[14] Assche DV, Chaves-Fraga D, Dimou A. KROWN: A Benchmark for RDF Graph Materialisation. In: Demartini G, Hose K, Acosta M, Palmonari M, Cheng G, Skaf-Molli H, et al., editors. The Semantic Web - ISWC 2024 - 23rd International Semantic Web Conference, Baltimore, MD, USA, November 11-15, 2024, Proceedings, Part III. vol. 15233 of Lecture Notes in Computer Science. Springer; 2024. p. 20-39. Available from: https://doi.org/10.1007/978-3-031-77847-6_2.

[15] Chaves-Fraga D, Endris KM, Iglesias E, Corcho O, Vidal ME. What are the parameters that affect the construction of a knowledge graph? In: On the Move to Meaningful Internet Systems: OTM 2019 Conferences: Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21–25, 2019, Proceedings. Springer; 2019. p. 695-713.

[16] Arenas-Guerrero J, Iglesias-Molina A, Chaves-Fraga D, Garijo D, Corcho O, Dimou A. Declarative generation of RDF-star graphs from heterogeneous data. Semantic Web. 2025;16(2):SW-243602.

[17] Arenas-Guerrero J, Espinoza-Arias P, Bernabé-Diaz JA, Deshmukh P, Sánchez-Fernández JL, Corcho O. An RML-FNML module for Python user-defined functions in Morph-KGC. SoftwareX. 2024;26:101709.

[18] Iglesias E, Vidal ME. Results for Knowledge Graph Creation Challenge 2024: SDM-RDFizer. In: Proceedings of the 5th International Workshop on Knowledge Graph Construction; 2024. .

[19] Freund M, Schmid S, Dorsch R, Harth A. Performance Results of FlexRML in the KGCW Challenge 2024. In: Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Hersonissos, Greece; 2024. .

[20] Scrocca M, Carenini A, Grassi M, Comerio M, Celino I. Not everybody speaks RDF: knowledge conversion between different data representations. In: Proceedings of the 5th International Workshop on Knowledge Graph Construction. CEUR Workshop Proceedings. vol. 3718; 2024. p. 1613-0073.

[21] Stadler C, Bin S. KGCW2024 Challenge Report: RDFProcessingToolkit. In: Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Hersonissos, Greece; 2024. .

[22] Debruyne C, Van Assche D. The conformance of an RML processor built from scratch to validate RML specifications and test cases. In: 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024). CEUR-WS. org; 2024. .

[23] Iglesias EA, Vidal ME. Results for Knowledge Graph Creation Challenge 2025: SDM-RDFizer. In: Proceedings of the 6th International Workshop on Knowledge Graph Construction; 2025. .

[24] Chaves-Fraga D, Priyatna F, Cimmino A, Toledo J, Ruckhaus E, Corcho O. GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain. Journal of Web Semantics. 2020;65:100596.