Iglesias-Molina, A. et al. (2023). The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. In: Payne, T.R., et al. The Semantic Web – ISWC 2023. ISWC 2023. Lecture Notes in Computer Science, vol 14266. Springer, Cham. https://doi.org/10.1007/978-3-031-47243-5_9

The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF

Ana Iglesias-Molina[1], Dylan Van Assche[2], Julián Arenas-Guerrero[1], Ben De Meester[2], Christophe Debruyne[3], Samaneh Jozashoori[4,5], Pano Maria[6], Franck Michel[7], David Chaves-Fraga[1,8], and Anastasia Dimou[8]

[1] Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
[2] IDLab, Dept. Electronics & Information Systems, Ghent University – imec, Belgium
[3] Montefiore Institute, University of Liège, Belgium
[4] metaphacts GmbH, Germany
[5] TIB - Leibniz Information Center for Science and Technology, Germany
[6] Skemu, Netherlands
[7] University Cote d'Azur, CNRS, Inria, I3S, France
[8] KU Leuven – Flanders Make@KULeuven – Leuven.AI, Belgium

Abstract. The Relational to RDF Mapping Language (R2RML) became a W3C Recommendation a decade ago. Despite its wide adoption, its potential applicability beyond relational databases was swiftly explored. As a result, several extensions and new mapping languages were proposed. They tackled not only more heterogeneous data sources, but also a wider range of limitations that surfaced as R2RML was applied in real-world use cases. Over the years, one of these languages, the RDF Mapping Language (RML), has gathered a large community of contributors, users, and compliant tools. However, so far, there has been no well-defined set of features for the mapping language, nor was there a consensus-marking ontology. Consequently, it has become challenging for non-experts to fully comprehend and utilize the full range of the language's capabilities. After three years of work, the W3C Community Group on Knowledge Graph Construction proposes a new specification for RML. This paper presents the new modular RML ontology and the accompanying SHACL shapes that complement the specification. We discuss the motivations and challenges that emerged when extending R2RML, the methodology we followed to design the new ontology while ensuring its backward compatibility with R2RML, and the novel features which increase its expressiveness. The new ontology consolidates the potential of RML, empowers practitioners to define mapping rules for constructing RDF graphs that were previously unattainable, and allows developers to implement systems in adherence with [R2]RML.
Resource type: Ontology

DOI: 10.5281/zenodo.7918478
URL: `http://w3id.org/rml/portal/`

Keywords: Declarative Language · R2RML · RML · Knowledge Graph.

1 Introduction

In 2012, the Relational to RDF Mapping Language (R2RML) [43] was released as a W3C Recommendation. The R2RML ontology [8] provides a vocabulary to describe how an RDF graph should be generated from data in a relational databases (RDB). Although R2RML gained wide adoption, its potential applicability beyond RDBs quickly appeared as a salient need [53,67,80,91].

Targeting the generation of RDF from heterogeneous data sources other than RDBs, several extensions [91,53,80] preserving R2RML's core structure were proposed. As R2RML and the growing number of extensions were applied in a wider range of use cases, more limitations became evident [31,80]. Consequently, these languages were further extended with different features, e.g., the description of input data sources or output RDF (sub)graphs [80,99], data transformations [46,51,70,73], support for RDF-star [52,94], etc. Over the years, the RDF Mapping Language (RML) has gathered a large community of contributors and users, and a plethora of systems [32,98] and benchmarks [37,39,63].

Until recently, there was no well-defined, agreed-upon set of features for the RML mapping language, nor was there a consensus-marking ontology covering the whole set of features. Consequently, it has become challenging for non-experts to fully comprehend this landscape and utilize all capabilities without investing a substantial research effort. Therefore, the W3C Community Group on Knowledge Graph Construction [3], with more than 160 members, has convened every two weeks to review the RML specification over the past three years.

In this paper, we present the new modular RML ontology and the accompanying SHACL shapes [65] that complement the specification. We discuss the motivations and challenges that emerged by extending R2RML, the methodology we followed to design the new ontology while ensuring its backward compatibility with R2RML, and the novel features which increase its expressiveness. The new RML ontology and specification is the result of an attempt to (i) address multiple use cases from the community [36] (ii) streamline and integrate various features proposed to support these use cases, and (iii) adopt agreed-upon design practices that make it possible to come up with a coherent, integrated whole consisting of a core ontology [22] and multiple feature-specific modules [20,23,24,26]. The presented ontology consolidates the potential of RML enabling the definition of mapping rules for constructing RDF graphs that were previously unattainable, and the development of systems in adherence with both R2RML and RML.

This paper is organized as follows: In Section 2, we present the relevant concepts of R2RML and RML. In Section 3, we outline the motivations that drive this work and the challenges we tackle. In Section 4, we describe the methodology employed to redesign the RML ontology while maintaining backward compatibility, and in Section 5 the modules introduced with the various features. In Section 6, we present the early adoption and potential impact, followed by related work in Section 7. We conclude the paper with a summary of the presented contributions and future steps in Section 8.

2 Background: R2RML

R2RML mapping rules (Listing 2) are grouped within `rr:TriplesMap` (line 1), which contain one `rr:LogicalTable`, one `rr:SubjectMap` and zero to multiple `rr:PredicateObjectMap`. The `rr:LogicalTable` (lines 2-3) describes the input RDB, while `rr:SubjectMap` (lines 4-5) specifies how the subjects of the triples are created. A `rr:PredicateObjectMap` (lines 6-9) generates the predicate-object pairs with one or more `rr:PredicateMap` (line 7) and one or more `rr:ObjectMap` (lines 8-9). Zero or more `rr:GraphMap`, which indicate how to generate named graphs, can be assigned to both `rr:SubjectMap` and `rr:PredicateObjectMap`. It is also possible to join `rr:LogicalTables` replacing `rr:ObjectMap` by `rr:RefObjectMap`, which uses the subject of another *Triples Map* indicated in `rr:parentTriplesMap` as the object of the triple. This join may have a condition to be performed, which is indicated using `rr:joinCondition`, `rr:child`, and `rr:parent`. *Subject Map*, *Predicate Map*, *Object Map*, and *Graph Map* are subclasses of `rr:TermMap`, which define how to generate RDF terms. *Term Maps* can be (i) *constant-valued*, i.e., always generating the same RDF term (line 7); (ii) *column-valued*, i.e., the RDF terms are directly obtained from cells of a column in the RDB (line 9); or (iii) *template-valued*, i.e., the RDF terms are composed from the data in columns and constant strings (line 5).

```
1  PERSON     , MARK, DATE
2  Duplantis  , 6.22, 02-25-2023
3  Guttormsen, 6.00, 03-10-2023
4  Vloon      , 5.91, 02-25-2023
```

Listing 1: *ATHLETES* table.

```
<#MarksTM> a rr:TriplesMap;                    1
 rr:logicalTable [                             2
  rr:tableName "ATHLETES" ];                   3
 rr:subjectMap [                               4
  rr:template ":{NAME}" ];                     5
 rr:predicateObjectMap [                       6
  rr:predicate :mark;                          7
  rr:objectMap [                               8
   rr:column "MARK" ] ] .                      9
```

Listing 2: R2RML mapping rules.

According to the R2RML specification, an R2RML processor is a system that, given a set of R2RML mapping rules and an input RDB, can construct RDF graphs. Therefore, an R2RML processor should have an SQL connection to the input RDB where the tables reside and a base IRI used to resolve the relative IRIs produced by the R2RML mapping rules.

3 Motivation and Challenges

In this section, we discuss the limitations of generalizing R2RML to construct RDF graphs from heterogeneous data sources and their impact on the ontology. We also consider the required extensions for the ontology to construct RDF graphs that were not possible before, e.g., RDF collections and containers or RDF-star [60]. Based on these limitations, we group the challenges in the follow-

ing high-level categories: data input and RDF output, schema and data transformations, collections and containers, and RDF-star.

Data Input & RDF Output. In R2RML, the desired RDF graph is constructed from tables residing in only one RDB. R2RML recommends to hard-code the connection to the RDB in the R2RML processor, hence, rules in a mapping document cannot refer to multiple input RDBs.

To date, a wide range of data formats and structures is considered beyond RDBs, such as CSV, XML, or JSON. These sources may be available locally or via web APIs, statically, or streaming. Thus, a flexible approach for constructing RDF graphs from a combination of these diverse inputs is desired [98]. The R2RML ontology needs to be extended to also describe what the data source is for each set of mapping rules, e.g., a NoSQL DB or a Web API, and what the data format is, e.g., CSV, JSON or XML. In addition, a per row iteration pattern is assumed for RDBs, but this may vary for other data formats.

RML [53] proposed how to describe heterogeneous data assuming originally that these data appear in local files and a literal value specifies the path to the local file. In parallel, xR2RML [80] proposed how to extend R2RML for the document-oriented MongoDB. A more concrete description of the data sources and their access, e.g., RDBs, files, Web APIs, etc. was later proposed [54], relying on well-known vocabularies to describe the data sources, e.g., DCAT [78], VOID [28], or SPARQL-SD [102] and further extended [99] to also describe the output RDF (sub)graphs. The description of `NULL` values [97], predetermined in RDBs but not in other data sources, has not been addressed yet.

Schema & Data Transformations. Integrating heterogeneous data goes beyond schema-level transformations, as it usually involves additional data-level transformations [71]. The R2RML ontology describes the schema transformations, i.e., the correspondences between the ontology and the data schema. It delegates data transformations and joins to the storage layer, by using operators in SQL queries. However, not all data formats can leverage similar operators, e.g., JSON does not have a formal specification to describe its data transformation, nor do all formats' operators cover the same data transformations, e.g., XPath offers a different set of operators compared to SQL. Moreover, there are cases in which such pre-processing is not possible, e.g., for streaming data. Thus, the R2RML ontology needs to be extended to describe such data transformations.

RML+FnO [45], R2RML-F [51], its successor FunUL [70] or D-REPR [101] are examples of the proposals providing support to data operations. However, only RML+FnO describes the transformation functions declaratively. RML+FnO has been well adopted in the community by being included in a number of RML-compliant engines [90,62,63,30] and RML+FnO-specific translation engines [89,69,68]. Nevertheless, a more precise definition to address ambiguities and a simplification of introduced (complex) constructs is needed.

Collections & Containers. RDF containers represent open sets of RDF terms, ordered (`rdf:Sequence`) or unordered (`rdf:Bag`, `rdf:Alt`). Their member terms are denoted with the `rdf:`$\_n$ properties[9]. RDF collections refer solely to type

[9] n is a strictly positive natural number denoting the nth element in that container.

`rdf:List` that represents a closed-ordered list of RDF terms. An RDF list is built using cons-pairs; the first cons-pair of a list refers to an element of that list with the `rdf:first` property, and the `rdf:rest` to the remainder list. All list elements should be traversed via `rdf:rest` until the empty list `rdf:nil`. Generating RDF collections and containers in R2RML, while possible, results in a cumbersome and limited task. A container's properties `rdf:_`n are typically generated only when a key in the form of a positive integer is yielded from the data source. By contrast, there is no elegant way to model a list's cons-pairs with R2RML if the list is of arbitrary length.

Due to the need for RDF containers and collections in several projects [79], e.g., both the Metadata Authority Description Schema [15] and W3C's XHTML Vocabulary [7] use RDF containers, and both OWL [103] and SHACL [72] use RDF collections. The xR2RML [80] vocabulary supported the generation of nested collections and containers within the same data source iteration (e.g., within one result among the results returned by the MongoDB database). Its vocabulary also allowed to change the iterator within a term map and yield nested collections and containers. By contrast, [50] provided terms for creating (nested) collections and containers from within an iteration (same row) and across iterations (across rows) and provided a property for retaining empty collections and containers. The ontology presented in [50] also provided directive for generating collections or containers whose members may have different term types, whereas [80]'s vocabulary provided support for one term type. The vocabulary of both approaches did not provide support for named collections and containers, nor the generation of these as subjects.

RDF-star. RDF-star [59] introduces the *quoted triple* term, which can be embedded in the subject or object of another triple. Quoted triples may be asserted (i.e., included in the graph) or not. RDF-star quickly gained popularity, leading to its adoption by a wide range of systems [4] (e.g., Apache Jena [29], Oxigraph [82]) and the formation of the RDF-star Working Group [5].

The inception of RDF-star came after R2RML. Therefore, R2RML only considered the generation of RDF. The principal challenge is the generation of quoted and asserted triples, which requires a dedicated extension. RML-star [52] and R2RML-star [94] are extensions of R2RML to construct RDF-star graphs, however, the latter comes with limitations and it is not backward compatible with R2RML. Our ontology includes the RML-star extension to enable the generation of RDF-star graphs, remaining backward compatible with R2RML.

4 Methodology

We followed the Linked Open Terms (LOT) methodology [85] to redesign the R2RML ontology, as well as to generalize and modularize it. The methodology includes four major stages: *Requirements Specification*, *Implementation*, *Publication*, and *Maintenance*. We describe below how we follow these steps to develop the RML ontology and the accompanying SHACL shapes.

Requirements. The requirements to build the RML ontology are mainly derived from three sources: (i) the legacy of the R2RML ontology, (ii) the scientific

publications which proposed different extensions [98,67], and (iii) the experience of the community of R2RML and RML to build upon their limitations. The latter has been gathered from GitHub issues [21] and summarized as *mapping challenges* [13]. The complete set of requirements for each module can be accessed from the ontology portal [25]. These requirements cover both the base needs and fine-grained features for generating triples with mapping rules. On the one hand, how to generate subjects, predicate, objects, datatypes, language tags, and named graphs in both a static (constant) and dynamic (from data sources) manner (RML-Core). On the other hand, the description and access of input data sources and target output data (RML-IO); RDF Collections and Containers to create lists from diverse terms (RML-CC); data transformation functions with their desired output and input parameters (RML-FNML); and quoting *Triples Maps* to create asserted and non-asserted RDF-star triples (RML-star).

Implementation. We build the RML ontology based on the requirements in a modular manner maintaining its backward compatibility with R2RML. We use a GitHub organization [2] to summarize issues and coordinate asynchronously.

Modularity. The ontology is composed of 5 modules: RML-Core, RML-IO, RML-CC, RML-FNML, and RML-star. We opt for a modular design to facilitate its development and maintenance, as each module can be adjusted independently without affecting the rest. This choice facilitates also its reuse and adoption, as RML processors can implement specific modules instead of the entire ontology.

Modeling. The modeling of each module is carried out independently. A version is drafted from the requirements and presented to the community. For this iteration step, we draft the proposal using ontology diagrams that follow the Chowlk notation [40], and some use cases with examples. Once it is agreed that the model is accurate and meets the requirements, the ontology is encoded.

Encoding. We encode the ontology using OWL [34] and its application profile using SHACL [72]. We deliberately choose to use both to distinguish between the model, which is described in OWL, and the constraints described as SHACL shapes. The latter allows to validate the mapping rules' correctness, depending on which modules are used. This way, RML processors can indicate which module they support and verify the mapping rules' compliance before executing them.

Backward compatibility. The new RML ontology is backwards compatible with the previous [R2]RML ontologies [17]. We first gather all terms affected from the RML-Core and RML-IO modules (the other modules only introduce new features), and define correspondences between the past and new resources. We identify two kinds of correspondences: (i) *equivalences* if a resource is used in the same manner and its semantics is not significantly changed (e.g., `rr:SubjectMap` is equivalent to `rml:SubjectMap`); and (ii) *replacements* if a resource is superseded by another one (e.g., `rr:logicalTable` is replaced by `rml:logicalSource`). A summary of these correspondences is available online [18] as well as a semantic version to enable automatic translation of mapping rules [17].

Evaluation. We evaluate the ontology with OOPS! [84] and check for inconsistencies using the HermiT reasoner. If all issues are solved, a module is deployed.

Publication. All modules of the RML ontology are managed and deployed independently from a separate GitHub repository, and published using a W3ID

Table 1: List of modules of the RML ontology.

Module	Description	Ontology
RML-Core	Schema Transformations	`http://w3id.org/rml/core`
RML-IO	Source and Target	`http://w3id.org/rml/io`
RML-CC	Collections and Containers	`http://w3id.org/rml/cc`
RML-FNML	Data Transformations	`http://w3id.org/rml/fnml`
RML-Star	RDF-star	`http://w3id.org/rml/star`

URL under the CC-BY 4.0 license. Each repository contains the ontology file, its documentation (created using Widoco [56]), requirements, associated SHACL shapes, and the module's specification. We follow a unified strategy for the resources' IRIs. The RML ontology resources use a single prefix IRI to make it convenient for users to convert their RML mappings to the new RML ontology, while clearly stating which module each resource belongs to. We publish the complete ontology at `http://w3id.org/rml`, and a summary of all modules with links to all their related resources (i.e. SHACL shapes, issues, specifications, etc.) is available at the ontology portal [25] and in Table 1.

Maintenance. To ensure that the ontology is updated with error corrections and new updates during its life cycle, the GitHub issue tracker of each module will be used to gather suggestions for additions, modifications, and deletions. We discuss every major modification asynchronously and in the W3C KG Construction Community Group meetings until they are agreed upon, which triggers another round of implementation and publication, leading to new releases.

5 Artifacts: Ontologies and Shapes

The RML ontology consists of 5 modules: (i) **RML-Core** (Section 5.1) describes the schema transformations, generalizes and refines the R2RML ontology, and becomes the basis for all other modules; (ii) **RML-IO** (Section 5.2) describes the input data and output RDF (sub)graphs; (iii) **RML-CC** (Section 5.3) describes how to construct RDF collections and containers; (iv) **RML-FNML** (Section 5.4) describes data transformations; and (v) **RML-star** (Section 5.5) describes how RDF-star can be generated. Fig. 1 shows an overview of all modules of the RML ontology and how they are connected. The modules build upon the RML-Core, which, in turn, builds upon R2RML, but are independent among one another. We illustrate each module by continuing the example in Section 2.

5.1 RML-Core: Schema Transformations

The RML-Core is the main module of the RML ontology, which generalizes and refines the R2RML ontology; all the other modules build on top of it. The RML-Core ontology consists of the same concepts as the R2RML ontology (`rml:TriplesMap`, `rml:TermMap`, `rml:SubjectMap`, `rml:PredicateMap`,

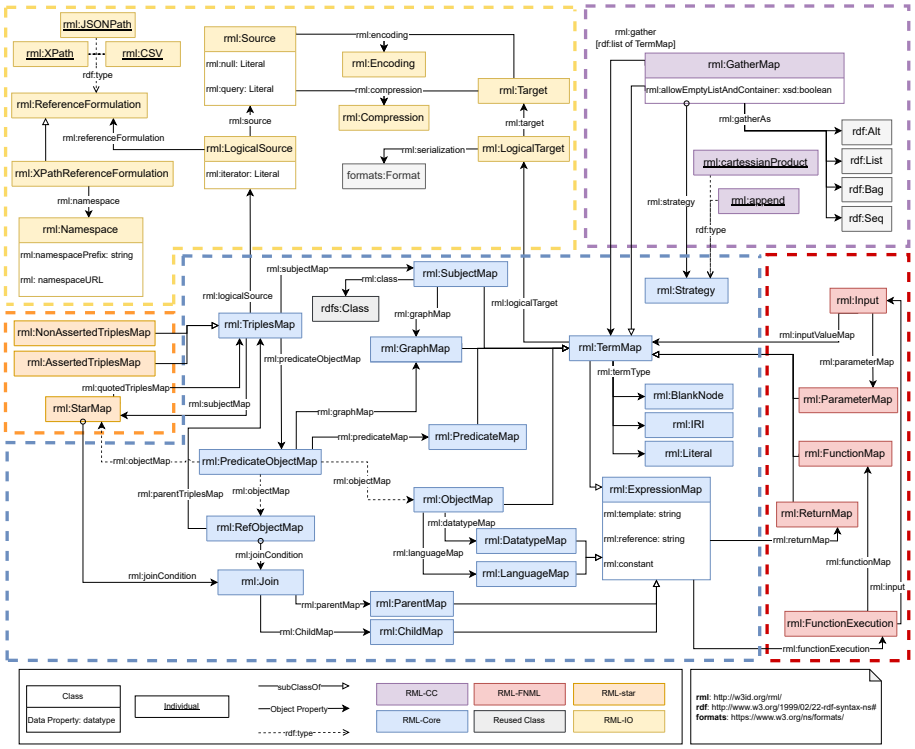

Fig. 1: RML ontology overview following the Chowlk diagram notation [40].

`rml:ObjectMap`, `rml:PredicateObjectMap`, and `rml:ReferencingObjectMap`), but redefines them to distinguish them from the R2RML counterparts.

RML-Core refines the R2RML ontology by introducing the concept of *Expression Map* `rml:ExpressionMap` (Listing 4, lines 6 & 9). An Expression Map is a mapping construct that can be evaluated on a data source to generate values during the mapping process, the so-called *expression values*. The R2RML specification allowed such mapping constructs (template-based, column-based or constant-based) which can only be applied to subject, predicate, object and named graph terms. In RML, the *Expression Map* can be a *template expression* specified with the property `rml:template`, a *reference expression* specified with the property `rml:reference`, or a *constant expression*, specified with the property `rml:constant`. A *Term Map* becomes a subclass of *Expression Map*.

With the introduction of the *Expression Map*, the language, term type, parent and child properties can be specified using any *Expression Map*, and not only a predefined type of expression. To achieve this, the following concepts are introduced as subclasses of the *Expression Map*: (i) `rml:LanguageMap`, whose shortcut `rml:language` can be used if it is a constant-valued *Expression Map*; (ii) `rml:DatatypeMap`, whose shortcut `rml:datatype` can be used if it is a constant-valued *Expression Map*; (iii) `rml:ParentMap`, whose shortcut `rml:parent` can be

used if it is a reference-valued *Expression Map*; and (iv) `rml:ChildMap`, whose shortcut `rml:child` can be used if it is a constant-valued *Expression Map*.

Listing 4 shows an example of a basic mapping to create RDF triples from the JSON file in Listing 3, and whose *Logical Source* is defined in Listing 5.

```
[ { "NAME": "Duplantis",
    "RANK": "1",
    "MARK": "6.22",
    "DATE": 02-25-2023},
  { "NAME": "Guttormsen",
    "RANK": "2",
    "MARK": "6.00",
    "DATE": 03-10-2023},
  { "NAME": "Vloon",
    "RANK": "3",
    "MARK": "5.91",
    "DATE": 02-25-2023} ]
```

Listing 3: Input JSON file with ranks and their specific date for athletes.

```
<#RankTriplesMap> a rml:TriplesMap;
  rml:logicalSource <#JSONSource>;
  rml:subjectMap <#RankSubjectMap>;
  rml:predicateObjectMap [ a rml:PredicateObjectMap;
    rml:predicate ex:rank;
    rml:objectMap [ a rml:ObjectMap, rml:ExpressionMap;
    rml:reference "$.RANK"; ] ] .

<#RankSubjectMap> a rml:SubjectMap, rml:ExpressionMap;
  rml:template "{$.NAME}" .
```

Listing 4: RML-Core example to generate a subject from a template, a predicate from a constant, and an object from a reference expression.

5.2 RML-IO: Source and Target

RML-IO complements RML-Core describing the input data sources and how they can be retrieved. To achieve this, RML-IO defines the *Logical Source* (with `rml:LogicalSource`) for describing the input data, and the *Source* (`rml:Source`) for accessing the data. The *Logical Source* specifies the grammar to refer to the input data via the *Reference Formulation* (`rml:ReferenceFormulation`). For instance, in Listing 5, the *Reference Formulation* is `JSONPath` (Line 4). RML-IO refers to a set of predefined *Reference Formulations* (`JSONPath, XPath, etc.`) but others can be considered as well. Besides the *Reference Formulation*, the *Logical Source* also defines how to iterate over the data source through the iteration pattern with the property `rml:iteration` (Line 5). In a *Triples Map*, the property `rml:logicalSource` specifies the *Logical Source* to use and should be specified once. The *Source* specifies how a data source is accessed by leveraging existing specifications; and indicates when values should be considered NULL (`rml:null`) and a query if needed (`rml:query`) e.g., SQL or SPARQL query. In a *Logical Source*, the property `rml:source` refers to exactly one *Source*.

Similarly, RML-IO includes the *Logical Target* (`rml:LogicalTarget`) and the *Target* (`rml:Target`) to define how the output RDF is exported. A *Logical Target* includes the properties `rml:serialization` to indicate in which RDF serialisation the output should be encoded, and `rml:target` to refer to exactly one *Target*. The *Logical Target* can be optionally specified in any *Term Map* e.g., *Subject* or *Graph Map*. The *Target* is similar to the *Source*, indicating how the output target can be accessed, and has 2 properties: (i) `rml:compression` to specify if the RDF output will be compressed and, if so, how e.g., GZip, and (ii) `rml:encoding` to define the encoding e.g., UTF-8.

Both *Source* and *Target* consider the re-use of existing vocabularies to incorporate additional features to access data, such as DCAT [78], SPARQL-SD [102], VoID [28], D2RQ [42], and CSVW [96]. Listing 5 shows an example of an RML mapping that describes a JSON file (Listing 3) with a *Logical Source* (lines 1-5). A *Logical Target* is also used to export the resulting triples to a Turtle file with GZip compression (lines 7-12). Since the *Logical Target* is specified within the subject (line 18), all triples with that subject are exported to this target.

```
1  <#JSONSource> a rml:LogicalSource;
2    rml:source [ a rml:Source, dcat:Distribution;
3      dcat:accessURL <file://ranks.json> ];
4    rml:referenceFormulation rml:JSONPath;
5    rml:iterator "$.[*]"; .
6
7  <#FileTarget> a rml:LogicalTarget;
8    rml:target [ a rml:Target, void:Dataset;
9      void:dataDump <file:///data/dump.ttl.gz>;
10     rml:compression comp:gzip;
11     rml:encoding enc:UTF-8 ];
12   rml:serialization formats:Turtle; .
13
14 <#RankTriplesMap> a rml:TriplesMap;
15   rml:logicalSource <#JSONSource>;
16   rml:subjectMap <#RankSubjectMap> .
17
18 <#RankSubjectMap> rml:logicalTarget <#FileTarget> .
```

Listing 5: Input data from the *ranked.json* file is described with DCAT. An output file in Turtle serialization with GZip compression is described with VoID.

5.3 RML-CC: Collections and Containers

As the RML Collections and Containers module is fundamentally new, the Community Group formulated a set of functional requirements that the RML Containers and Collections specification should meet: One should be able to: (i) collect values from one or more *Term Maps*, including multi-valued *Term Maps*; (ii) have control over the generation of empty collections and containers; (iii) generate nested collections and containers; (iv) group different term types;

(v) use a generated collection or container as a subject; and (vi) assign an IRI or blank node identifier to a collection or container. Based on these requirements, the RML-CC module was introduced which consists of a new concept, the *Gather Map* `rml:GatherMap`, two mandatory properties (`rml:gather` and `rml:gatherAs`) and two optional properties. Even though the module is limited with respect to its ontology terms, significant effort is expected for the RML processors to support it. Thus, we decided on keeping it as a separate module.

We specified the *Gather Map* (`rml:GatherMap`) as a *Term Map* with 2 mandatory properties: (i) `rml:gather` to specify the list of *Term Maps* used for the generation of a collection or container, and (ii) `rml:gatherAs` to indicate what is generated (one of the `rdf:List`, `rdf:Bag`, `rdf:Seq`, and `rdf:Alt`). The `rml:gather` contains any type of *Term Map* including *Referencing Term Maps* (to use the subjects generated by another *Triples Map*) which are treated as multi-valued *Term Maps*. Other properties were defined with default values to facilitate the use of this extension: `rml:allowEmptyListAndContainer` and `rml:strategy`. By default, a *Gather Map* shall not yield empty containers and collections; the predicate `rml:allowEmptyListAndContainer` must be set to true to preserve them[10]. Also, by default, the values of multiple multi-valued *Term Maps* will be appended from left to right; `rml:append` is the default `rml:strategy`. Alternatively, the `rml:cartesianProduct` strategy that instructs to carry out a Cartesian product between the terms generated by each *Term Map*. The `rml:strategy` renders the vocabulary `extensible`; RML implementations may propose their own strategies for generating collections and containers from a list of multi-valued term maps.

Listing 6 demonstrates the support for 4 of the aforementioned requirements: the collection of values from a multi-valued *Term Map*, the generation of a named collection, and the collection as a subject. It generates a subject that will be related to a list via `ex:contains`. The values of that list are collected from a multi-valued *Term Map* generating IRIs from the names and generates the following RDF: `:Ranking23 ex:contains (:Duplantis :Guttormsen :Vloon).`

```
<#RankingListTM> a rml:TriplesMap;
  rml:logicalSource <#JSONSource>;
  rml:subjectMap [ rml:constant :Ranking23 ];
  rml:predicateObjectMap [
    rml:predicate ex:contains;
    rml:objectMap [
      rml:gather (
        [ rml:template "{$.*.NAME}"; rml:termType rml:IRI ] );
      rml:gatherAs rdf:List; ] ] .
```

Listing 6: The use of a *Gather Map* to generate a list of terms. The RDF collection `:Ranking23` will be generated using the name data reference.

If a *Gather Map* is an empty *Expression Map*, a new blank node is created for the head node of each generated collection or container (the first cons-pair

[10] E.g., an empty list could explicitly represent that the number 1 has no prime factors.

in the case of a collection). Conversely, when providing a template, constant or reference, the head node is assigned the generated IRI or blank node identifier. If unchecked, this may lead to the generation of collections or containers that share the same head node, which we refer to as ill-formed. Therefore, the specification details the behavior that a processor must adopt: when a gather map creates a named collection or container, it must first check whether a named collection or container with the same head node IRI or blank node identifier already exists, and if so, it must append the terms to the existing one.

5.4 RML-FNML: Data Transformations

The RML-FNML module enables the declarative evaluation of data transformation functions defined using the Function Ontology (FnO) [46] in RML. Thus, the data transformation functions in RML are independent of specific processors. Functions and Executions are described with FnO, while FNML declares the evaluation of FnO functions in terms of specific data sources. The *evaluation* of a function is defined through a *Function Execution* (`rml:FunctionExecution`), where a *Function Map* (`rml:FunctionMap`) defines the function. The input values' definitions are provided through *Term Maps* using *Inputs* (`rml:Input`), which in turn include *Parameter Maps* (`rml:ParameterMap`) referring to *Function Parameters* defined by FnO. The *Function Execution*'s output is declared using a *Return Map* (`rml:ReturnMap`) and referred to by the `rml:return` property, enabling the reference to a specific output of a function's multiple outputs.

```
1  <#RankTriplesMap> a rml:TriplesMap;
2    rml:logicalSource <#JSONSource>;
3    rml:subjectMap  <#RankSubjectMap>;
4    rml:predicateObjectMap [
5      rml:predicate ex:date;
6      rml:objectMap [
7        rml:functionExecution <#Execution>;
8        rml:return ex:dateOut ] ] .
9
10 <#Execution> a rml:FunctionExecution;
11   rml:function ex:parseDate;
12   rml:input [ a rml:Input;
13     rml:parameter ex:valueParam;
14     rml:inputValueMap [ rml:reference "$.DATE" ] ] ,
15     [ a rml:Input;
16     rml:parameter ex:dateFormatParam;
17     rml:inputValueMap [ rml:constant "MM-DD-YYYY" ] ] .
```

Listing 7: The function `ex:parseDate` parses the referenced `DATE` value using the `"MM-DD-YYYY"` pattern, and returns a parsed date.

Listing 7 shows the use date formatting function. Within an *Object Map*, the *Function Execution* (Line 7) and type of *Return* (Line 8) are defined. The *Function Execution* describes which *Function* is used (Line 11) and its two *Inputs*: the data reference (Lines 12-14) and the output date format (Lines 15-17).

5.5 RML-star

The building block of RML-star [52] is the *Star Map* (`rml:StarMap`). A *Star Map* can be defined in a *Subject* or an *Object Map*, generating quoted triples in the homonymous positions of the output triples. The *Triples Map* generating quoted triples is connected to a *Star Map* via the object property `rml:quotedTriplesMap`. Quoted *Triple Maps* specify whether they are asserted (`rml:AssertedTriplesMap`) or non-asserted (`rml:NonAssertedTriplesMap`). Listing 8 uses an *Asserted Triples Map* (lines 1-5) to generate triples of the mark of some athletes, annotated using a *Star Map* with the date on which the marks were accomplished (lines 7-11).

```
1  <#QuotedRankTM> a rml:AssertedTriplesMap;
2    rml:logicalSource <#JSONSource>;
3    rml:subjectMap <#RankSubjectMap>;
4    rml:predicateObjectMap [ rml:predicate :mark;
5      rml:objectMap [ rml:reference "$.MARK"; ] ] .
6
7  <#DateTM> a rml:TriplesMap;
8    rml:logicalSource <#JSONSource> ;
9    rml:subjectMap [ rml:quotedTriplesMap <#RankTriplesMap> ] ;
10   rml:predicateObjectMap [ rml:predicate :date;
11     rml:objectMap [ rml:reference "$.DATE"; ] ] .
```

Listing 8: The <#QuotedRankTM> generates asserted triples that are also quoted by `rml:quotedTriplesMap` property from <#DateTM> .

6 Early Adoption and Potential Impact

Over the years, the RML mapping language has gathered a large community of contributors and users, a plethora of systems were developed [32,98], benchmarks were proposed [37,39,63], and tutorials were performed [9,10,11,12,16,100].

During the last decade, many initiatives have used the different extensions of R2RML which contributed to the modular RML ontology to construct RDF graphs from heterogeneous data for e.g., COVID-19-related data [81,88,93], biodiversity [66,83,27], streaming data analysis and visualisation [44,49], social networks' data portability [48], social media archiving [77], supply chain data integration [47], public procurement [92], agriculture [35], federated advertisement profiles [76]. These extensions were also incorporated in services, e.g., Chimera [57], Data2Services [14], InterpretME [41], and even the Google Enterprise Knowledge Graph to construct and reconcile RDF [6].

As the modules of the RML ontology take shape, an increasing number of systems embrace its latest version. The RMLMapper [62], which was so far the RML reference implementation, currently supports the RML-Core and RML-IO modules. The SDM-RDFizer [63,64] and Morph-KGC [30], two broadly-used RML processors, already integrate the generation of RDF-star with the RML-star module in their systems. Additionally, Morph-KGC also supports transfor-

mation functions using the RML-FNML module. The adoption of RML was facilitated by YARRRML [61], a human-friendly serialization of RML. Yatter [38] is a YARRRML translator that already translates this serialization to the RML-Core, RML-star, RML-IO and RML-FNML modules. The increasing number of systems that support different modules illustrate the benefits of the modular approach of the RML ontology: each system implements a set of modules, without the necessity of offering support for the complete ontology, while different use cases can choose a system based on their modules' support.

As an increasing number of systems support the RML ontology proposed in this paper, several real-world use cases already adopted the ontology as well. NORIA [95] extends RMLMapper in their MASSIF-RML [86] system, implemented at Orange, to construct RDF graphs for anomaly detection and incident management. InteGraph [74] uses RML to construct RDF graphs in the soil ecology domain and CLARA [19] to construct RDF-star for educational modules, both using Morph-KGC. At the European level, two governmental projects use RML to integrate their data into RDF graphs. In the transport domain, the European Railway Agency constructs RDF from the Register of Infrastructure data, distributed in XML, using RML mapping rules [87] and taking advantage of the RML-IO module. The Public Procurement Data Space [58] is an ongoing project that integrates procurement data, distributed in various formats, from all EU member states using the e-Procurement Ontology [1], and mapping rules in RML with the RML-Core and RML-star modules on the roadmap.

The RML ontology and SHACL shapes are maintained by the W3C Knowledge Graph Construction Community Group, and part of the larger International Knowledge Graph Construction community. This community will continue to maintain these resources and a call for systems to incorporate the new specification and ontology will be launched. The aim is to have at least two reference implementations for each module in RML systems by the end of 2023.

7 Related Work

A large amount of mapping languages have been proposed to enable the construction of RDF graphs from different data sources. Apart from the RDF-based languages that considerably influence the RML ontology (see Section 3), we highlight here the popular alternatives that rely on the syntax of query (SPARQL-Generate [75], SPARQL-Anything [33]), constraint (e.g., ShExML [55]) or data-serialisation (e.g., D-REPR [101]) languages to construct RDF graphs.

SPARQL-Generate [75] leverages the expressive power of the SPARQL query language and extends it with additional clauses to describe the input data. It offers a relatable way to RML for handling the input data: it supports a wide range of data sources, describes their access and defines an iterator and reference formulations to describe input data. While SPARQL-Generate describes input data and its access, it does not consider the specification of target formats as the new RML ontology does. SPARQL-Generate supports collection and containers, but they can only be placed as objects; embedded collections and containers are not allowed. Last, despite developed over Apache Jena, which already supports

RDF-star and SPARQL-star, the GENERATE clause proposed by SPARQL-Generate, does not support RDF-star graph construction at the moment.

SPARQL-Anything [33] introduces *Facade-X* to override the SERVICE clause as its input data description. It implements all SPARQL and SPARQL-star features, including the generation of RDF-star graphs. However, the construction of well-formed collections and containers with the CONSTRUCT clause is limited. To overcome this, SPARQL-Anything proposes a bespoke function `fx:bnode` that ensures that the same blank node identifiers are returned for the same input. Hence, while blank nodes are addressed, the generation of lists remains complex. Both SPARQL-Generate and SPARQL-Anything, offer limited support for data transformations, as they are bounded to the ones already provided by their corresponding implementations. While SPARQL allows custom functions, these are implementation-dependent. The addition of new data transformations declaratively, as the new RML ontology proposes, is not possible.

Despite the expressive power of these languages, the systems that implement them need to provide a complete support for SPARQL and extend the language with new clauses or modify the semantics of existing ones to support the construction of RDF graphs. The modular approach presented for the RML ontology allows having a set of basic features to be implemented by the systems, without forcing support for all the language, and ensures the long-term sustainability, as new modules of the ontology can be proposed without affecting current ones.

8 Conclusions and Future Steps

We present the RML ontology as a community-driven modular redesign of R2RML and its extensions to generate RDF graphs from heterogeneous data sources. Our work is driven by the limitations of R2RML and the extensions proposed over the years. We presented our motivation for following a modular design, backwards compatible with R2RML. We discussed how each module was designed accompanied by its SHACL shapes, addressing the identified challenges.

The quick adoption of the RML ontology by some of its most used systems, and the number of initiatives and companies that have already incorporated RML, creates a favorable ecosystem for the adoption of RML as the standard for generating RDF graphs from heterogeneous data. The modular design allows us to easily adjust the adequate module with future requirements following an agile methodology. A thorough versioning system will be enabled to keep track of the new versions and badges will be provided for systems to indicate which modules and versions of these modules they support.

As future steps, the community is willing to initiate the process of turning this resource into a W3C Recommendation. Hence, a *Final Community Group Report* will be published with all the resources presented in this paper, so the SW community can start providing feedback on the specifications to finally, draft a W3C Working Group charter. From a technical perspective, we want to develop further use cases to ensure a thorough validation of the new implementations. Finally, test-cases for each module and validation with SHACL shapes will also be further refined to provide an exhaustive validation resource.

References

1. eProcurement Ontology (ePO). https://github.com/OP-TED/ePO, accessed May 9, 2023
2. GitHub Organization of the Knowledge Graph Construction W3C Community Group. https://www.github.com/kg-construct/, accessed May 9, 2023
3. Knowledge Graph Construction Community Group. https://www.w3.org/community/kg-construct/, accessed May 9, 2023
4. RDF-star Implementations. https://w3c.github.io/rdf-star/implementations.html, accessed May 9, 2023
5. RDF-star Working Group. https://www.w3.org/groups/wg/rdf-star, accessed May 9, 2023
6. Run an entity reconciliation job from the Google Cloud console. https://cloud.google.com/enterprise-knowledge-graph/docs/entity-reconciliation-console, accessed May 9, 2023
7. XHTML Vocabulary. https://www.w3.org/1999/xhtml/vocab (2010), accessed May 9, 2023
8. R2RML: RDB to RDF Mapping Language Schema. https://www.w3.org/ns/r2rml# (2012), accessed May 9, 2023
9. Tutorial: Generating and Querying (Virtual) Knowledge Graphs from Heterogeneous Data Sources. https://oeg-dataintegration.github.io/kgc-tutorial-2019 (2019), accessed May 9, 2023
10. Tutorial: How to build a knowledge graph. https://2019.semantics.cc/satellite-events/how-build-knowledge-graph (2019), accessed May 9, 2023
11. Tutorial: How to build large knowledge graphs efficiently (LKGT). https://stiinnsbruck.github.io/lkgt/ (2020), accessed May 9, 2023
12. Tutorial: Knowledge Graph Construction using Declarative Mapping Rules. https://oeg-dataintegration.github.io/kgc-tutorial-2020 (2020), accessed May 9, 2023
13. Knowledge Graph Construction Open Challenges. https://w3id.org/kg-construct/workshop/2021/challenges.html (2021), accessed May 9, 2023
14. Data2Services: RML Transformations. https://d2s.semanticscience.org/docs/d2s-rml (2022), accessed May 9, 2023
15. Metadata Authority Description Schema. https://www.loc.gov/standards/mads/ (2022), accessed May 9, 2023
16. Tutorial: Knowledge Graph Construction. https://w3id.org/kg-construct/costdkg-eswc-tutorial (2022), accessed May 9, 2023
17. Backwards Compatibility . http://w3id.org/rml/bc (2023), accessed May 9, 2023
18. Backwards Compatibility Portal. https://w3id.org/rml/portal/backwards-compatibility.html (2023), accessed May 9, 2023
19. Clara Project. https://gitlab.univ-nantes.fr/clara/pipeline (2023), accessed May 9, 2023
20. RML-CC Ontology: Collections and Containers. https://w3id.org/rml/cc (2023), accessed May 9, 2023
21. RML Core issues. https://github.com/kg-construct/rml-core/issues (2023), accessed May 9, 2023
22. RML-Core Ontology: Generic Mapping Language for RDF. https://w3id.org/rml/core (2023), accessed May 9, 2023

23. RML-FNML Ontology: Functions. `https://w3id.org/rml/fnml` (2023), accessed May 9, 2023
24. RML-IO Ontology: Source and Target. `https://w3id.org/rml/io` (2023), accessed May 9, 2023
25. RML Ontology Portal. `http://w3id.org/rml/portal/` (2023), accessed May 9, 2023
26. RML-star Ontology. `https://w3id.org/rml/star` (2023), accessed May 9, 2023
27. Aisopos, F., Jozashoori, S., Niazmand, E., Purohit, D., Rivas, A., Sakor, A., Iglesias, E., Vogiatzis, D., Menasalvas, E., Gonzalez, A.R., et al.: Knowledge graphs for enhancing transparency in health data ecosystems. Semantic Web Journal (2023)
28. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing Linked Datasets with the VoID Vocabulary. Interest Group Note, World Wide Web Consortium (2011), `https://www.w3.org/TR/void/`
29. Apache Software Foundation: Apache Jena (2021), `https://jena.apache.org`
30. Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M.S., Corcho, O.: Morph-KGC: Scalable knowledge graph materialization with mapping partitions. Semantic Web (2022). https://doi.org/10.3233/SW-223135
31. Arenas-Guerrero, J., Alobaid, A., Navas-Loro, M., Pérez, M.S., Corcho, O.: Boosting Knowledge Graph Generation from Tabular Data with RML Views. In: Proceedings of the 20th Extended Semantic Web Conference. Springer International Publishing
32. Arenas-Guerrero, J., Scrocca, M., Iglesias-Molina, A., Toledo, J., Pozo-Gilo, L., Doña, D., Corcho, O., Chaves-Fraga, D.: Knowledge Graph Construction with R2RML and RML: An ETL System-based Overview. In: Proceedings of the 2nd International Workshop on Knowledge Graph Construction. vol. 2873. CEUR Workshop Proceedings (2021), `http://ceur-ws.org/Vol-2873/paper11.pdf`
33. Asprino, L., Daga, E., Gangemi, A., Mulholland, P.: Knowledge graph construction with a façade: A unified method to access heterogeneous data sources on the web. ACM Trans. Internet Technol. (2022). https://doi.org/10.1145/3555312
34. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A., et al.: OWL Web Ontology Language. W3C Recommendation, World Wide Web Consortium (2004), `https://www.w3.org/TR/owl-ref/`
35. Bilbao-Arechabala, S., Martinez-Rodriguez, B.: A practical approach to cross-agri-domain interoperability and integration. In: 2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS). pp. 1–6. IEEE (2022)
36. Chaves, D., Osborne, F., Heyvaert, P., Michel, F., Iglesias, A., Riccitelli, D., Xiao, G., García, H., Rojas, J., Vander Sande, M., Jozashoori, S., Volpini, A., De Meester, B., Otaku, Maria, P., Shigapov, R., Alexiev, V.: kg-construct/use-cases: v1.0 (2023). https://doi.org/10.5281/zenodo.7907172
37. Chaves-Fraga, D., Endris, K.M., Iglesias, E., Corcho, O., Vidal, M.E.: What are the Parameters that Affect the Construction of a Knowledge Graph? In: Proceedings of the Confederated International Conferences. pp. 695–713. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-33246-4_43
38. Chaves-Fraga, D., Gonzalez, M., Lopez, L., Doña, D., Guerrero, J.A., Ahmed, S., Corcho, O.: oeg-upm/yatter: v1.1.0 (2023). https://doi.org/10.5281/zenodo.7898764
39. 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 **65**, 100596 (2020)

40. Chávez-Feria, S., García-Castro, R., Poveda-Villalón, M.: Chowlk: from uml-based ontology conceptualizations to owl. In: The Semantic Web: 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29–June 2, 2022, Proceedings. pp. 338–352. Springer (2022)
41. Chudasama, Y., Purohit, D., Rohde, P.D., Gercke, J., Vidal, M.E.: InterpretME: A Tool for Interpretations of Machine Learning Models Over Knowledge Graphs. Submitted to Semantic Web Journal (2023), https://www.semantic-web-journal.net/system/files/swj3404.pdf
42. Cyganiak, R., Bizer, C., Garbers, J., Maresch, O., Becker, C.: The D2RQ Mapping Language. Tech. rep., FU Berlin, DERI, UCB, JP Morgan Chase, AGFA Healthcare, HP Labs, Johannes Kepler Universität Linz (2012), http://d2rq.org/d2rq-language
43. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Recommendation, World Wide Web Consortium (2012), http://www.w3.org/TR/r2rml/
44. De Brouwer, M., Bonte, P., Arndt, D., Vander Sande, M., Heyvaert, P., Dimou, A., Verborgh, R., De Turck, F., Ongenae, F.: Distributed continuous home care provisioning through personalized monitoring & treatment planning. In: Companion Proceedings of the Web Conference 2020. ACM (Apr 2020). https://doi.org/10.1145/3366424.3383528
45. De Meester, B., Dimou, A., Verborgh, R., Mannens, E.: An Ontology to Semantically Declare and Describe Functions. In: Extended Semantic Web Conference, P&D. pp. 46–49. Springer International Publishing (2016)
46. De Meester, B., Seymoens, T., Dimou, A., Verborgh, R.: Implementation-independent function reuse. Future Generation Computer Systems **110**, 946–959 (2020). https://doi.org/10.1016/j.future.2019.10.006
47. De Mulder, G., De Meester, B.: Implementation-independent Knowledge Graph Construction Workflows using FnO Composition. In: Third International Workshop on Knowledge Graph Construction (2022), https://ceur-ws.org/Vol-3141/paper4.pdf
48. De Mulder, G., De Meester, B., Heyvaert, P., Taelman, R., Verborgh, R., Dimou, A.: PROV4ITDaTa: Transparent and Direct Transfer of Personal Data to Personal Stores. In: Proceedings of The Web Conference (2021). https://doi.org/10.1145/3442442.3458608
49. De Paepe, D., Vanden Hautte, S., Steenwinckel, B., Moens, P., Vaneessen, J., Vandekerckhove, S., Volckaert, B., Ongenae, F., Van Hoecke, S.: A Complete Software Stack for IoT Time-Series Analysis that Combines Semantics and Machine Learning—Lessons Learned from the Dyversify Project. Applied Sciences **11**(24), 11932 (Dec 2021). https://doi.org/10.3390/app112411932
50. Debruyne, C., McKenna, L., O'Sullivan, D.: Extending R2RML with support for RDF collections and containers to generate MADS-RDF datasets. In: Kamps, J., Tsakonas, G., Manolopoulos, Y., Iliadis, L.S., Karydis, I. (eds.) 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. Lecture Notes in Computer Science, vol. 10450, pp. 531–536. Springer (2017). https://doi.org/10.1007/978-3-319-67008-9_42
51. Debruyne, C., O'Sullivan, D.: R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In: Proceedings of the 9th Workshop on Linked Data on the Web. vol. 1593. CEUR Workshop Proceedings (2016), http://ceur-ws.org/Vol-1593/article-13.pdf

52. Delva, T., Arenas-Guerrero, J., Iglesias-Molina, A., Corcho, O., Chaves-Fraga, D., Dimou, A.: RML-star: A Declarative Mapping Language for RDF-star Generation. In: International Semantic Web Conference, ISWC, P&D. vol. 2980. CEUR Workshop Proceedings (2021), http://ceur-ws.org/Vol-2980/paper374.pdf
53. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: Bizer, C., Heath, T., Auer, S., Berners-Lee, T. (eds.) Proceedings of the 7th Workshop on Linked Data on the Web. vol. 1184. CEUR Workshop Proceedings (2014), http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf
54. Dimou, A., Verborgh, R., Vander Sande, M., Mannens, E., Van de Walle, R.: 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). https://doi.org/10.1145/2814864.2814873
55. García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J.E., Lovelle, J.M.C.: ShExML: improving the usability of heterogeneous data mapping languages for first-time users. PeerJ Computer Science **6**, e318 (2020)
56. Garijo, D.: Widoco: a wizard for documenting ontologies. In: 6th International Semantic Web Conference, Vienna, Austria. pp. 94–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_9
57. Grassi, M., Scrocca, M., Carenini, A., Comerio, M., Celino, I.: Composable Semantic Data Transformation Pipelines with Chimera. In: Proceedings of the 4th International Workshop on Knowledge Graph Construction. CEUR Workshop Proceedings (2023)
58. Guasch, C., Lodi, G., Van Dooren, S.: Semantic knowledge graphs for distributed data spaces: The public procurement pilot experience. In: The Semantic Web–ISWC 2022: 21st International Semantic Web Conference, Virtual Event, October 23–27, 2022, Proceedings. pp. 753–769. Springer (2022)
59. Hartig, O.: Foundations of RDF* and SPARQL* (An Alternative Approach to Statement-Level Metadata in RDF). In: Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web. CEUR Workshop Proceedings, vol. 1912 (2017)
60. Hartig, O., Champin, P.A., Kellogg, G., Seaborne, A.: RDF-star and SPARQL-star. W3C Final Community Group Report (2021), https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html
61. 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. pp. 213–217. Springer (2018)
62. Heyvaert, P. and De Meester, B. et al.: RMLMapper (2022), https://github.com/RMLio/rmlmapper-java
63. Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., Vidal, M.E.: SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs. In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management, CIKM. pp. 3039–3046. Association for Computing Machinery (2020). https://doi.org/10.1145/3340531.3412881
64. Iglesias, E., Vidal, M.E.: SDM-RDFizer-Star (2022), https://github.com/SDM-TIB/SDM-RDFizer-Star
65. Iglesias-Molina, A., Assche, D.V., Arenas-Guerrero, J., Meester, B.D., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D., Dimou, A.: Rml ontology and shapes (May 2023). https://doi.org/10.5281/zenodo.7918478

66. Iglesias-Molina, A., Chaves-Fraga, D., Priyatna, F., Corcho, O.: Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings. In: SWAT4HCLS. pp. 1–10 (2019)
67. Iglesias-Molina, A., Cimmino, A., Ruckhaus, E., Chaves-Fraga, D., García-Castro, R., Corcho, O.: An ontological approach for representing declarative mapping languages. Semantic Web pp. 1–31 (2022). https://doi.org/10.3233/sw-223224
68. Jozashoori, S.: Semantic data integration and knowledge graph creation at scale (2023). https://doi.org/10.15488/13537
69. Jozashoori, S., Chaves-Fraga, D., Iglesias, E., Vidal, M.E., Corcho, O.: FunMap: Efficient Execution of Functional Mappings for Knowledge Graph Creation. In: Proceedings of the 19th International Semantic Web Conference, ISWC. pp. 276–293. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-62419-4_16
70. Junior, A.C., Debruyne, C., Brennan, R., O'Sullivan, D.: FunUL: A Method to Incorporate Functions into Uplift Mapping Languages. In: Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services. p. 267–275. Association for Computing Machinery (2016). https://doi.org/10.1145/3011141.3011152
71. Knoblock, C.A., Szekely, P.: Exploiting semantics for big data integration. AI Magazine **36**(1), 25–38 (2015). https://doi.org/10.1609/aimag.v36i1.2565
72. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL) (2017), `https://www.w3.org/TR/shacl/`
73. Kyzirakos, K., Savva, D., Vlachopoulos, I., Vasileiou, A., Karalis, N., Koubarakis, M., Manegold, S.: GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. Journal of Web Semantics **52-53**, 16–32 (2018). https://doi.org/10.1016/j.websem.2018.08.003
74. Le Guillarme, N., Thuiller, W.: A Practical Approach to Constructing a Knowledge Graph for Soil Ecological Research. bioRxiv (2023). https://doi.org/10.1101/2023.03.02.530763
75. Lefrançois, M., Zimmermann, A., Bakerally, N.: A SPARQL Extension for Generating RDF from Heterogeneous Formats. In: Proceedings of the 14th Extended Semantic Web Conference. pp. 35–50. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-58068-5_3
76. Lieber, S., De Meester, B., Verborgh, R., Dimou, A.: EcoDaLo: Federating Advertisement Targeting with Linked Data. Lecture Notes in Computer Sciencie, vol. 12378, pp. 87–103. Springer, Cham (Oct 2020). https://doi.org/10.1007/978-3-030-59833-4_6
77. Lieber, S., Van Assche, D., Chambers, S., Messens, F., Geeraert, F., Birkholz, J.M., Dimou, A.: BESOCIAL: A Sustainable Knowledge Graph-Based Workflow for Social Media Archiving. In: Further with Knowledge Graphs. pp. 198–212. IOS Press (2021)
78. Maali, F., Erickson, J.: Data Catalog Vocabulary (DCAT). W3C Recommendation, World Wide Web Consortium (2014), `https://www.w3.org/TR/vocab-dcat/`
79. McKenna, L., Bustillo, M., Keefe, T., Debruyne, C., O'Sullivan, D.: Development of an RDF-Enabled Cataloguing Tool. 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. Lecture Notes in Computer Science, vol. 10450, pp. 612–615. Springer (2017). https://doi.org/10.1007/978-3-319-67008-9_55

80. Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: Translation of Relational and Non-relational Databases into RDF with xR2RML. In: Monfort, V., Krempels, K., Majchrzak, T.A., Turk, Z. (eds.) WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015. pp. 443–454. SciTePress (2015). https://doi.org/10.5220/0005448304430454
81. Michel, F., Gandon, F., Ah-Kane, V., Bobasheva, A., Cabrio, E., Corby, O., Gazzotti, R., Giboin, A., Marro, S., Mayer, T., et al.: Covid-on-the-web: Knowledge graph and services to advance covid-19 research. In: The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II 19. pp. 294–310. Springer (2020)
82. Pellissier Tanon, T.: Oxigraph (2023). https://doi.org/10.5281/zenodo.7749949
83. Pérez, A.Á., Iglesias-Molina, A., Santamaría, L.P., Poveda-Villalón, M., Badenes-Olmedo, C., Rodríguez-González, A.: Eboca: Evidences for biomedical concepts association ontology. In: Knowledge Engineering and Knowledge Management: 23rd International Conference, EKAW 2022, Bolzano, Italy, September 26–29, 2022, Proceedings. pp. 152–166. Springer (2022)
84. Poveda-Villalón, M., Gómez-Pérez, A., Suárez-Figueroa, M.C.: OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. International Journal on Semantic Web and Information Systems **10**(2), 7–34 (2014). https://doi.org/10.4018/ijswis.2014040102
85. Poveda-Villalón, M., Fernández-Izquierdo, A., Fernández-López, M., García-Castro, R.: LOT: An industrial oriented ontology engineering framework. Engineering Applications of Artificial Intelligence **111**, 104755 (2022). https://doi.org/10.1016/j.engappai.2022.104755
86. Ranaivoson, M., Tailhardat, L., Chabot, Y., Troncy, R.: SMASSIF-RML: a Semantic Web stream processing solution with declarative data mapping capability based on a modified version of the RMLMapper-java tool and extensions to the StreamingMASSIF framework. (Mar 2023), `https://github.com/Orange-OpenSource/SMASSIF-RML`
87. Rojas, J.A., Aguado, M., Vasilopoulou, P., Velitchkov, I., Van Assche, D., Colpaert, P., Verborgh, R.: Leveraging semantic technologies for digital interoperability in the European railway domain. In: The Semantic Web–ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings 20. pp. 648–664. Springer (2021)
88. Sakor, A., Jozashoori, S., Niazmand, E., Rivas, A., Bougiatiotis, K., Aisopos, F., Iglesias, E., Rohde, P.D., Padiya, T., Krithara, A., et al.: Knowledge4COVID-19: A semantic-based approach for constructing a COVID-19 related knowledge graph from various sources and analyzing treatments' toxicities. Journal of Web Semantics **75**, 100760 (2023)
89. Samaneh Jozashoori, E.I., Vidal, M.E.: Dragoman (2022), `https://github.com/SDM-TIB/Dragoman`
90. Şimşek, U., Kärle, E., Fensel, D.: RocketRML-A NodeJS implementation of a use-case specific RML mapper. In: International Workshop on Knowledge Graph Building (2019)
91. Slepicka, J., Yin, C., Szekely, P., Knoblock, C.A.: KR2RML: An Alternative Interpretation of R2RML for Heterogeneous Sources. In: Proceedings of the 6th International Workshop on Consuming Linked Data. vol. 1426. CEUR Workshop Proceedings (2015), `http://ceur-ws.org/Vol-1426/paper-08.pdf`
92. Soylu, A., Corcho, O., Elvesæter, B., Badenes-Olmedo, C., Blount, T., Yedro Martínez, F., Kovacic, M., Posinkovic, M., Makgill, I., Taggart, C., et al.:

TheyBuyForYou platform and knowledge graph: Expanding horizons in public procurement with open linked data. Semantic Web **13**(2), 265–291 (2022)
93. Steenwinckel, B., Vandewiele, G., Rausch, I., Heyvaert, P., Colpaert, P., Simoens, P., Dimou, A., De Turck, F., Ongenae, F.: Facilitating COVID-19 Meta-analysis Through a Literature Knowledge Graph. In: Proc. of 19th International Semantic Web Conference (ISWC) (2020)
94. Sundqvist, L.: Extending VKG Systems with RDF-star Support (2022), https://ontop-vkg.org/publications/2022-sundqvist-rdf-star-ontop-msc-thesis.pdf
95. Tailhardat1, L., Chabot, Y., Troncy, R.: Designing NORIA: a Knowledge Graph-based Platform for Anomaly Detection and Incident Management in ICT Systems. In: Proceedings of the 4th International Workshop on Knowledge Graph Construction. CEUR Workshop Proceedings (2023)
96. Tennison, J., Kellogg, G., Herman, I.: Generating RDF from Tabular Data on the Web. W3C Recommendation, World Wide Web Consortium (2015), https://www.w3.org/TR/csv2rdf/
97. Toussaint, E., Guagliardo, P., Libkin, L., Sequeda, J.: Troubles with Nulls, Views from the Users. Proceedings of the VLDB Endowment **15**(11), 2613–2625 (2022). https://doi.org/10.14778/3551793.3551818
98. Van Assche, D., Delva, T., Haesendonck, G., Heyvaert, P., De Meester, B., Dimou, A.: Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review. Journal of Web Semantics **75**, 100753 (2023). https://doi.org/10.1016/j.websem.2022.100753
99. Van Assche, D., Haesendonck, G., De Mulder, G., Delva, T., Heyvaert, P., De Meester, B., Dimou, A.: Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation. In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds.) Web Engineering. Lecture Notes in Computer Science, vol. 12706, pp. 337–352. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-74296-6_26
100. Van Herwegen, J., Heyvaert, P., Taelman, R., De Meester, B., Dimou, A.: Tutorial: Knowledge Representation as Linked Data: Tutorial. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. pp. 2299–2300 (2018)
101. Vu, B., Pujara, J., Knoblock, C.A.: D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF. In: Proceedings of the 10th International Conference on Knowledge Capture. p. 189–196. Association for Computing Machinery (2019). https://doi.org/10.1145/3360901.3364449
102. Williams, G.: SPARQL 1.1 Service Description. W3C Recommendation, World Wide Web Consortium (2013), https://www.w3.org/TR/sparql11-service-description/
103. Williams, G.T.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation, World Wide Web Consortium (2012), https://www.w3.org/TR/owl2-overview/