Morph-KGC^{star}: Declarative generation of RDF-star graphs from heterogeneous data

Julián Arenas-Guerrero [a,*,**], Ana Iglesias-Molina [a,**], David Chaves-Fraga [a,b,c], Daniel Garijo [a], Oscar Corcho [a] and Anastasia Dimou [b,c]

[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
E-mails: julian.arenas.guerrero@upm.es, ana.iglesiasm@upm.es, daniel.garijo@upm.es, oscar.corcho@upm.es
[b] *Declarative Languages and Artificial Intelligence Group, KU Leuven, Belgium*
E-mails: david.chaves@kuleuven.be, anastasia.dimou@kuleuven.be
[c] *Flanders Make, DTAI-FET, Belgium*

Abstract. RDF-star has been proposed as an extension of RDF to annotate statements with triples. Libraries and graph stores have started adopting RDF-star, but the generation of RDF-star data remains largely unexplored. To allow generating RDF-star from heterogeneous data, RML-star was proposed as an extension of RML. However, no implementation has been developed so far that implements the RML-star specification. In this work, we present Morph-KGC^{star}, which extends the Morph-KGC materialization engine to generate RDF-star datasets. We validate Morph-KGC^{star} by running test cases derived from the N-Triples-star syntax tests and we apply it to two real-world use cases from the biomedical and open science domains. We compare the performance of our approach against other RDF-star generation methods (SPARQL-Anything), showing that Morph-KGC^{star} scales better for large input datasets, but it is slower when processing multiple smaller files.

Keywords: Knowledge Graphs, RDF-star, RML-star, Data Integration

1. Introduction

RDF-star [1] was proposed as an extension of RDF [2] to annotate statements and, thus, make statements about other statements (also known as reification [3]). RDF-star extends the RDF's conceptual data model and concrete syntaxes by providing a compact alternative to other reification approaches, such as *standard reification* [4] or *singleton properties* [5]. Following the uptake of the initial version of RDF-star, the W3C RDF-DEV Community Group[1] recently released a W3C Final Community Group Report [6] and the RDF-star Working Group[2] has recently been formed to extend related W3C Recommendations.

Even though several libraries and graph stores have already adopted RDF-star[3], the generation of RDF-star graphs remains largely unexplored. RDF graphs are often generated from heterogeneous semi-structured data, e.g., data in CSV, XML or JSON formats, etc. To generate RDF graphs, mapping languages are used to specify how RDF terms and triples can be generated from these data. The syntax of these mapping languages are either custom or repurposed.

*Corresponding author. E-mail: julian.arenas.guerrero@upm.es.
**The authors contributed equally to this work.
[1] https://www.w3.org/groups/cg/rdf-dev
[2] https://www.w3.org/groups/wg/rdf-star
[3] https://w3c.github.io/rdf-star/implementations

The syntax of custom mapping languages is designed for generating RDF graphs, such as the W3C Recommendation R2RML [7], for generating RDF from data in relational databases, and its extensions for heterogeneous data, e.g., RDF Mapping Language (RML) [8] or xR2RML [9]. Alternatively, mapping languages may repurpose an existing syntax proposed for other scopes, e.g., based on the query languages SPARQL [10], such as SPARQL-Generate [11] or SPARQL-Anything [12, 13], or on the constraints language ShEx [14], such as ShExML [15].

Mapping languages focused so far on the generation of RDF graphs, but the emergence of RDF-star brings a new challenge. Depending on the underlying syntax, the mapping languages employ different mechanisms to support the generation of RDF graphs. On the one hand, SPARQL-based mapping languages can take advantage of the SPARQL-star extension [6] as long as their adjustments to the syntax are not affected and the implementation they are based on allows it. For instance, SPARQL-Anything is built on top of Apache Jena [16], which was recently extended to support RDF-star and SPARQL-star. On the other hand, dedicated mapping languages require an extension both over their syntax and their implementations. In our previous work, we proposed an extension over RML, namely RML-star [17], to describe how RDF-star graphs can be generated from heterogeneous semi-structured data. However, to the best of our knowledge, no RML-star processor has been implemented so far.

In this work, we present Morph-KGC^{star}, an open source implementation of RML-star that generates RDF-star graphs. The contributions of this paper are: (i) an updated release of RML-star, compliant with the latest RDF-star specification; (ii) an algorithm to process RML-star and generate RDF-star knowledge graphs; (iii) its implementation as an extension of Morph-KGC [18]; (iv) a validation of the algorithm and its implementation based on test and use cases; (v) a comlause and Apaon of our proposal against other approaches for generating reified RDF (standard reification and singleton properties) in terms of the generation time; and (vi) a comparison with SPARQL-Anything, a SPARQL-based language to generate RDF-star graphs.

The rest of the paper is structured as follows: Section 2 introduces background terminology and concepts. Section 3 describes and compares different approaches to generate statements about statements with RML and RML-star. Section 4 introduces our solution, Morph-KGC^{star}, and explains how RDF-star datasets can be generated using RML-star mappings. Section 5 presents the validation process we followed to ensure the quality of our approach. Section 6 briefly describes related work and finally Section 7 concludes the paper and outlines future work lines.

2. Background

In this section we briefly describe RDF-star, the target data model of our proposal, and RML, the mapping language that we extend to generate RDF-star graphs.

RDF-star [1] was proposed as an extension of RDF to concisely annotate statements represented as RDF triples. RDF-star captures the notion of "*quoted triple*", which in the concrete syntaxes are enclosed using "«" and "»". An RDF-star triple can be placed in the subject or object of an RDF triple and can be recursive, i.e., a quoted triple can contain in turn other quoted triples. For example, the RDF-star triple «`:Angelica :jumps "4.80"`» `:date "2022-03-21".` semantically describes that Angelica scored a specific height on a specific date. RDF-star triples that are an element of the RDF-star graph are known as *asserted triples*. In our example, «`:Angelica :jumps "4.80"`» is a quoted triple, which can also be asserted if included in the RDF-star graph.

RML [8] extends the W3C Recommendation R2RML [7] to declaratively define how to generate RDF graphs from heterogeneous data (not only relational databases, but also data in CSV, JSON, XML, etc.). Mapping rules in RML are encoded as a set of rules that describe how the triples of the RDF graph should be generated from the input data, usually following the schema provided by an ontology or network of ontologies. An RML mapping document is a set of `rr:TriplesMap`, each of them containing one `rml:LogicalSource`, one `rr:SubjectMap`, and from zero to multiple `rr:PredicateObjectMap`. The `rr:SubjectMap` declares how the subject of the triples are generated and it also indicates its class, using the property `rr:class`. A `rr:PredicateObjectMap` contains one or more `rr:PredicateMap` to define the predicates of the triples and, in a similar way, one or more `rr:ObjectMap` that declare how the objects should be generated. Subject maps and predicate object maps can have from zero to multiple `rr:GraphMap`, which describe how to generate named graphs (if generated). When a join between logical sources is needed, `rr:ObjectMap` is replaced by `rr:RefObjectMap`, which uses the subject maps of a triples map (`rr:parentTriplesMap`) to generate the

objects of the triples. A join condition between the triples maps involved in a referencing object map can be declared using the properties `rr:joinCondition`, `rr:child` and `rr:parent`. Subject, predicate, object and graph maps are `rr:TermMap`, which define a function to generate the RDF terms. Term maps can be constant (always generating the same RDF term), reference (the RDF terms are directly obtained from a data field) or template (the RDF terms are composed from multiple data fields and constant strings) valued.

3. Statements about Statements in Mapping Rules

Making statements about statements in RDF posed a challenge almost since the inception of RDF. Indeed, the W3C RDF Primer [19] already included a description of the standard reification approach. Other alternatives were proposed over the years, such as singleton properties [5], RDF$^+$ [20], and more recently, RDF-star [1].

This section describes popular reification approaches and shows how they can be used in RML and RML-star with a running example. Standard reification and singleton properties are considered in Section 5, showing that Morph-KGC*star* does not add any overhead in the time required to generate the RDF-star triples compared to them.

We illustrate each reification alternative with a running example that uses the data shown in Listing 1. It contains CSV data related to pole vault: the vaulter (`PERSON`), the height of the jump (`MARK`), the date when the jump was performed (`DATE`) and an identifier of the jump (`ID`). The running example represents that a person jumped some height on a specific date, i.e., it adds the metadata about "date" to the statement "a person jumped some height".

```
ID , DATE       , MARK , PERSON
1  , 2022-03-21 , 4.80 , Angelica
2  , 2022-03-19 , 4.85 , Katerina
```

Listing 1: Contents of the logical source `:marks` in CSV format.

3.1. Reification with RML

Two popular reification approaches exist: standard reification and singleton properties. These approaches use strategies that add metadata to triples without additional constructs (e.g., named graphs [3]). They can be used with RML without any further modification. RML mapping rules enable the generation of blank nodes (required for standard reification) and dynamically generated predicates (required for singleton properties).

Standard Reification [19] was proposed in the W3C RDF Primer [19]. It assigns statements to unique identifiers (typically blank nodes) typed with `rdf:Statement` and described using the properties `rdf:subject`, `rdf:predicate` and `rdf:object`. This way, the unique identifier representing the statement can be further annotated with additional statements. Listing 4 shows an example of standard reification for the data in Listing 1, created with the RML mapping rules in Listing 2. This mapping creates blank nodes in the subject with the `ID` data field, typed with `rdf:Statement`; and has three predicate object maps to generate the `rdf:subject`, `rdf:predicate`, `rdf:object` of the triples and a predicate object map to annotate the statements with `:date`.

Singleton Properties [5]. This approach uses unique predicates linked with `rdf:singletonPropertyOf` to the original predicate. This unique predicate can then be annotated as the subject of additional statements. Listing 5 shows the reified triples for the data in Listing 1 created with the RML mapping rules in Listing 3. It uses a singleton property dynamically generated with the `ID` data field for the property `:jumps`, annotated with `:date`.

```
<#TM> a rr:TriplesMap ;
  rml:logicalSource :marks ;
  rr:subjectMap [
    rml:reference "ID" ;
    rr:termType rr:BlankNode ;
    rr:class rdf:Statement ] ;
  rr:predicateObjectMap [
    rr:predicate rdf:subject ;
    rr:objectMap [
      rr:template ":{PERSON}" ] ] ;
  rr:predicateObjectMap [
    rr:predicate rdf:predicate ;
    rr:object :jumps ] ;
  rr:predicateObjectMap [
    rr:predicate rdf:object ;
    rr:objectMap [
      rml:reference "MARK" ] ] ;
  rr:predicateObjectMap [
    rr:predicate :date ;
    rr:objectMap [
      rml:reference "DATE" ] ] .
```

Listing 2: Example RML mapping using standard reification that transforms data in Listing 1.

```
_:1 rdf:type  rdf:Statement .
_:1 rdf:subject :Angelica .
_:1 rdf:predicate :jumps .
_:1 rdf:object "4.80" .
_:1 :date     "2022-03-21" .
_:2 rdf:type  rdf:Statement .
_:2 rdf:subject :Katerina .
_:2 rdf:predicate :jumps .
_:2 rdf:object "4.85" .
_:2 :date     "2022-03-19" .
```

Listing 4: RDF triples generated by the mapping in Listing 2.

```
<#TM> a rr:TriplesMap ;
  rml:logicalSource :marks ;
  rr:subjectMap [
    rr:template ":{PERSON}" ] ;
  rr:predicateObjectMap [
    rr:predicateMap [
     rr:template ":jumps#{ID}" ] ;
    rr:objectMap [
      rml:reference "MARK" ] ] .

<#TM-SP> a rr:TriplesMap ;
  rr:logicalSource :marks ;
  rr:subjectMap [
    rr:template ":jumps#{ID}" ] ;
  rr:predicateObjectMap [
    rr:predicate rdf:singletonPropertyOf;
    rr:object :jumps ] ;
  rr:predicateObjectMap [
    rr:predicate :date ;
    rr:objectMap [
      rml:reference "DATE" ] ] .
```

Listing 3: Example RML mapping using a singleton property that transforms data in Listing 1.

```
:Angelica :jumps#1      "4.80" .
:jumps#1 :date          "2022-03-21" .
:jumps#1 rdf:singletonPropertyOf :jumps .
:Katerina :jumps#2      "4.85" .
:jumps#2 :date          "2022-03-19" .
:jumps#2 rdf:singletonPropertyOf :jumps .
```

Listing 5: RDF triples generated by the mapping in Listing 3.

3.2. Reification with RML-star

In a previous work [17], we proposed RML-star (Figure 1) as an extension of RML to generate RDF-star graphs. RML-star adds a new kind of term map, the `rml:StarMap`, that allows using triples maps to generate quoted triples. Following the RDF-star data model, star maps can only be used in subject and object maps. Star maps use the property `rml:quotedTriplesMap` to refer to the triples map that generates the quoted triples. This referenced triples map will also generate asserted triples, since it is a `rr:TriplesMap`. To enable the generation of quoted triples without asserting them, RML-star introduces `rml:NonAssertedTriplesMap` as a subclass of `rr:TriplesMap`. Non-asserted triples maps can be referred by `rml:quotedTriplesMap` to generate quoted triples, but they will be ignored when generating asserted triples.

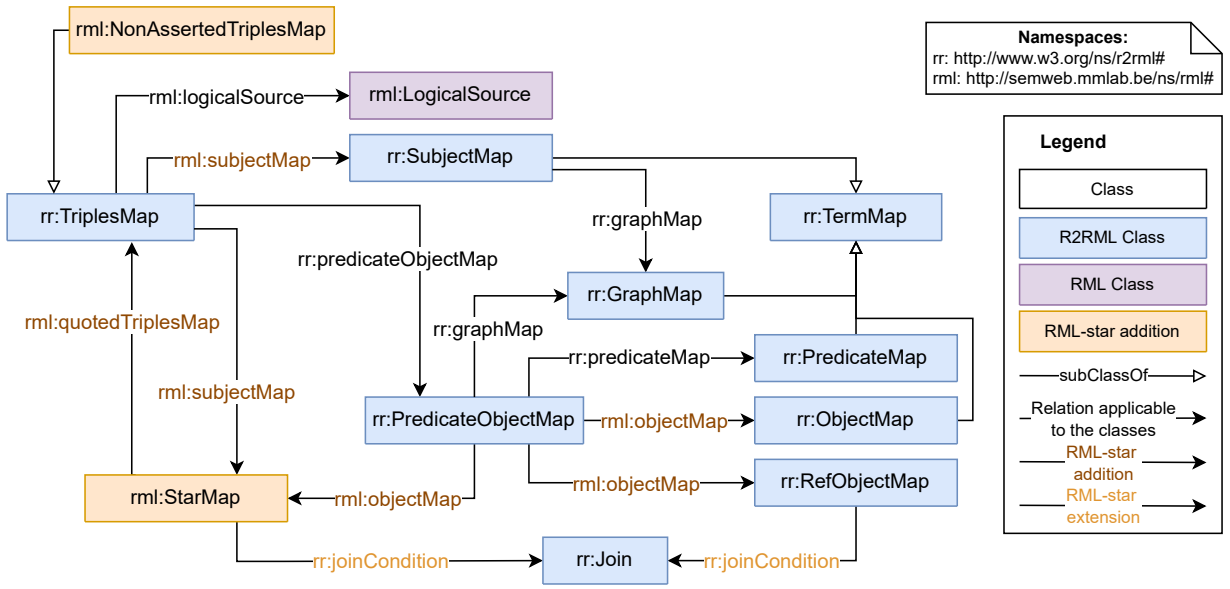

Fig. 1. The RML-star extension (represented using the Chowlk notation [21]). Orange classes and dark orange object properties show the additions to the RML ontology, light orange object properties represent extensions (i.e., modifications in the domain and/or range).

The RML-star specification [22] provides a complete description of the language, it is published as a W3C Draft Community Report, and it is maintained by the W3C Knowledge Graph Construction Community Group[4]. Both, the language and the specification are kept up to date reflecting the modifications in RDF-star. For instance, the latest RML-star releases update the term "embedded" to "quoted", according to the modifications in RDF-star. This update renamed the property `rml:embeddedTriplesMap` to `rml:quotedTriplesMap`. An example of an RML-star mapping rule for the data in Listing 1 is in Listing 6 which generates the RDF-star triples in Listing 7. The mapping rules use a non-asserted triples map (`<#innerTM>`) within the subject map of a triples map (`<#outerTM>`) which annotates quoted triples with `:date`.

```
<#innerTM>
  a rml:NonAssertedTriplesMap ;
  rml:logicalSource :marks ;
  rml:subjectMap [
    rr:template ":{PERSON}" ] ;
  rr:predicateObjectMap [
    rr:predicate :jumps ;
    rml:objectMap [
      rml:reference "MARK" ] ] .
```

```
<#outerTM>
  a rr:TriplesMap ;
  rml:logicalSource :marks ;
  rml:subjectMap [
    rml:quotedTriplesMap <#innerTM> ] ;
  rr:predicateObjectMap [
    rr:predicate :date ;
    rml:objectMap [
      rml:reference "DATE" ] ] .
```

Listing 6: Example RML-star mapping that transforms data in Listing 1.

```
<< :Angelica :jumps "4.80" >> :date "2022-03-21" .
<< :Katerina :jumps "4.85" >> :date "2022-03-19" .
```

Listing 7: RDF-star triples generated by the mapping in Listing 6.

[4]https://www.w3.org/community/kg-construct/

4. Morph-KGC*star*

In this section we describe Morph-KGC*star*. First, we address the materialization of RDF-star knowledge graphs with RML-star and provide an algorithm to generate the RDF-star triples of a mapping rule. Then, we describe our implementation and its features.

4.1. Materialization with RML-star

The materialization of an RML-star mapping rule is presented in Algorithm 1. An RML-star processor generates the output dataset of an RML-star document by applying Algorithm 1 to each mapping rule in the document. The mapping rules of a triples map to be processed by the algorithm are obtained by iterating over its predicate object maps, predicate maps, object maps and graph maps, so that only one subject, predicate, object and graph maps are processed at a time. Note that the R2RML specification[5] recommends processing triples maps in this way.

There are three types of term maps in RML-star that need to be differentiated for materialization: simple, referencing and star maps. Handling simple and referencing term maps is already considered in R2RML and RML materialization procedures that are well reported in the literature [7, 23]. Algorithm 1 covers them in *lines 5-6, 12-15 & 21* and more details of their materialization can be found in the W3C R2RML Recommendation.

Processing RML-star to generate RDF-star triples requires to additionally process non-asserted triples maps and star maps. A mapping rule in RML-star resembles a binary tree in which the left and right children are given by the mapping rules referenced by star maps in the subject and object respectively. This way, Algorithm 1 traverses the tree of mapping rules in post order: first, the left subtree (given by the star map in the subject) of the current mapping rule, then the right subtree (given by the star map in the object), and finally the current mapping rule is processed for generating the quoted triples. The last step also materializes the asserted triples and adds them to the output RDF-star dataset. Hereinafter we refer to the mapping rule in the root of the tree as the *outermost* mapping rule, and the rest as *inner* mapping rules. We use *level of nesting* to refer to the depth of a mapping rule in the tree.

Non-asserted triples maps must not generate asserted triples (i.e., the triples must not be added to the output RDF-star graph). This entails that the mapping rules within a non-asserted triples maps must only be processed when generating quoted triples. Algorithm 1 uses the `nestLevel` argument to keep track of the level of nesting which is being processed, with 0 referring to the outermost mapping rule. When a mapping rule within a non-asserted triples map is in the outermost level of nesting, it is discarded by Algorithm 1 (*lines 2-3*) as the asserted triples should not be created. If `nestLevel` is not 0, the generated triples will be quoted and the mapping rule should be processed.

Star maps can occur in both, the subject and object positions (*lines 7-10 & 16-19* respectively). Before generating the triples, the logical sources involved in the star map (a star map involves two triples maps) must be joined. In this way, the terms for the quoted triples and the annotation triple are generated from the same joint logical source, complying with the provided join condition. To achieve this, the parent triples map is retrieved from the mapping rules (*lines 8 & 17*), and the logical sources of both triples maps are merged into a joint logical source (*lines 9 & 18*). When the logical sources of the triples maps are the same and no join condition is provided *lines 9 & 18* have no effect. Given that they are the same, any of the original logical sources can be used as the joint logical source.

As star maps may lead to nested rules, processors should deal with any level of nesting. Considering the recursive nature of RML-star, the materialization of RDF-star graphs must also be implemented recursively, a significant challenge compared to RML. Algorithm 1 recursively calls `materializeMappingRule` (*lines 10 & 19*) passing the joint mapping rule (i.e., with the joint logical source) and increasing `nestLevel`, as a deeper level of nesting will be processed. In this way, the triples generated by the inner mapping rule will be quoted in the subject or object of the triples generated by the mapping rule at the current level of nesting.

So far we have only considered the generation of triples. However, quads can also be generated with graph maps. In RDF-star, quads are never quoted. However, in RML-star, triples maps are never restricted from having a graph map (i.e., inner triples map can also have a graph map). To prevent the generation of *quoted quads* in RML-star, graph maps must only be processed in the outermost mapping rule (i.e., the level of nesting in which triples or quads

[5] https://www.w3.org/TR/r2rml/#generated-triples

Algorithm 1 Materialization of an RML-star rule

```
1: procedure MATERIALIZEMAPPINGRULE(m, M, nestLevel = 0)
2:     if ISNONASSERTED(m) and nestLevel == 0 then
3:         return
4:     end if
5:     if ISSIMPLETERMMAP(m.SM) then
6:         subjects ← MATERIALIZETERMMAP(m.SM)
7:     else if ISSTARTERMMAP(m.SM) then
8:         m_parent ← GETMAPPINGRULE(m.SM, M)
9:         m_joint ← JOINMAPPINGRULES(m, m_parent)
10:        subjects ← MATERIALIZEMAPPINGRULE(m_joint, M, nestLevel + 1)
11:    end if
12:    if ISSIMPLETERMMAP(m.OM) then
13:        objects ← MATERIALIZETERMMAP(m.OM)
14:    else if ISREFTERMMAP(m.OM) then
15:        objects ← MATERIALIZEREFTERMMAP(m.OM, M)
16:    else if ISSTARTERMMAP(m.OM) then
17:        m_parent ← GETMAPPINGRULE(m.OM, M)
18:        m_joint ← JOINMAPPINGRULES(m, m_parent)
19:        objects ← MATERIALIZEMAPPINGRULE(m_joint, M, nestLevel + 1)
20:    end if
21:    predicates ← MATERIALIZETERMMAP(m.PM)
22:    if nestLevel == 0 then
23:        if HASGM(m) then
24:            namedGraphs ← MATERIALIZETERMMAP(m.GM)
25:            return CREATEQUADS(subjects, predicates, objects, namedGraphs)
26:        else
27:            return CREATETRIPLES(subjects, predicates, objects)
28:        end if
29:    else if nestLevel > 0 then
30:        return CREATESTARTRIPLES(subjects, predicates, objects)
31:    end if
32: end procedure
```

are asserted) and ignored otherwise. *Lines 22-25* of Algorithm 1 process graph maps when `nestLevel` is 0 and a graph map is provided, generating quads. If the outermost mapping rule has no graph map, RDF-star triples are added to the default graph of the output dataset (*lines 26-27*). When an inner mapping rule is being processed, the generated triples must be quoted (*lines 29-30*).

4.2. The RML-star Engine Morph-KGC^{star}

Morph-KGC [18] is an R2RML and RML compliant materialization engine implemented in Python and using Pandas [24] for data manipulation (i.e., through tables). Morph-KGC^{star} extends Morph-KGC to process RML-star and generate RDF-star graphs. Morph-KGC^{star} uses SQLAlchemy [25] to access relational databases. In this way, many popular database management systems are supported. In addition, it allows a wide range of tabular data sources powered by Pandas (CSV, Parquet, ORC, etc.) and hierarchical files (JSON and XML), which can also be accessed remotely. Morph-KGC^{star} enables the generation of RDF-star graphs from all of these data formats using RML-star. Figure 2 shows an overview of Morph-KGC^{star}.

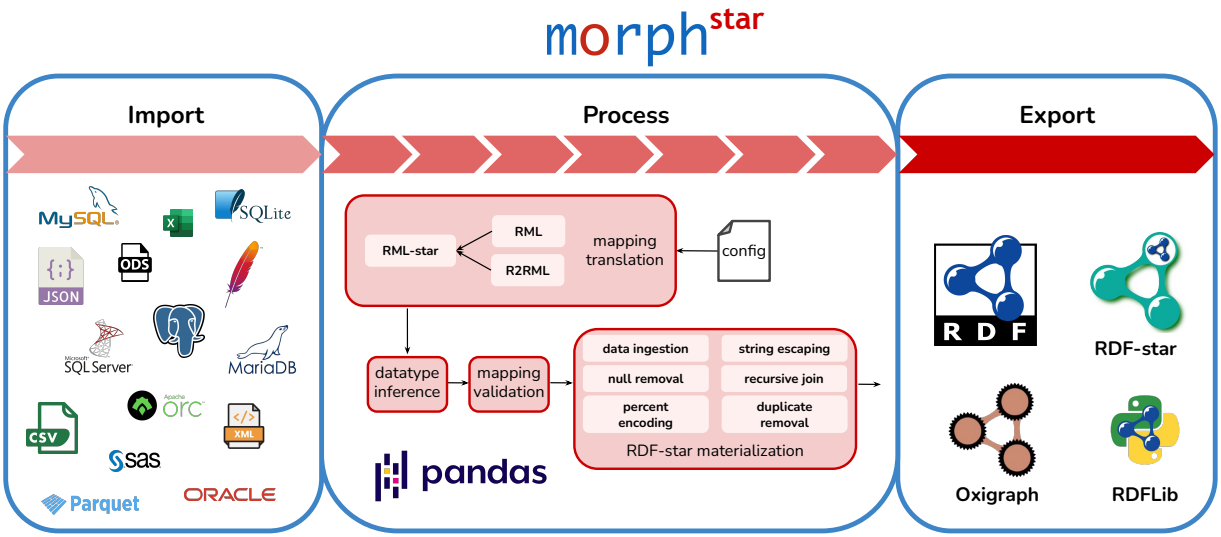

Fig. 2. Overview of Morph-KGC*star*.

There are two different ways of exporting RDF-star datasets in Morph-KGC*star*. The first option is to generate a file with the dataset in the N-Triples-star or the N-Quads-star serializations. This can be done by executing the engine from the command line. The other alternative is to use Morph-KGC*star* as a library and create an Oxigraph[6] store populated with RDF-star triples. We integrated Morph-KGC*star* with Oxigraph, as Morph-KGC only integrated originally with RDFLib [26], that at the time of writing does not support RDF-star. This new integration allows generating RDF-star knowledge graphs with Morph-KGC*star* and exploit them with Oxigraph entirely with Python.

Backward compatibility with R2RML and RML is ensured (Figure 2), because these mapping languages are subsets of RML-star. If a set of mapping rules is provided to Morph-KGC*star* in the R2RML or RML languages, they will be translated to RML-star. This translation step allows the engine to work with a common representation for all mapping rules. Morph-KGC*star* also allows completing the datatypes of literal term maps for relational databases[7].

Morph-KGC*star* uses tables internally to manipulate data. Dataframes are created for tabular data sources (e.g., relational databases or CSV files). For hierarchical data files, a DataFrame is created after evaluating the `rml:iterator`. Processing RML-star in Morph-KGC*star* resembles the nested relational model [27], in which the logical sources of deeply nested mapping rules correspond to tables and their join conditions define the relations between them. The engine performs the joins locally along with typical operations in RDF graph materialization, such as percent encoding or duplicate removal.

The source code of Morph-KGC*star* is maintained on GitHub[8] and the engine is distributed as a PyPi package[9]. The development of the engine is under continuous integration using GitHub Actions and the RML-star, RML and R2RML test cases. Every release of the engine is also stored in Zenodo [28]. Morph-KGC*star* is available under the Apache 2.0 License and its documentation is licensed under CC BY-SA 4.0 and available online[10].

The number of triplestores that now support RDF-star (e.g., GraphDB, Apache Jena, or Stardog) evidences its popularity and adoption by the community. However, RDF-star needs to be generated before it is exploited. The widespread use of declarative mappings [23] and the current lack of systems to generate RDF-star will contribute to the impact of Morph-KGC*star* in the Semantic Web community. We expect the system to become the reference implementation of RML-star and that it will open new lines of research, such as the optimization of the generation

[6] https://oxigraph.org/pyoxigraph
[7] https://www.w3.org/TR/r2rml/#natural-mapping
[8] https://github.com/oeg-upm/morph-kgc/releases/tag/2.0.0
[9] https://pypi.org/project/morph-kgc/
[10] https://morph-kgc.readthedocs.io

of RDF-star knowledge graphs. Thus, users and practitioners will benefit from this tool, having a sustainable way of creating RDF-star graphs and avoiding ad-hoc scripting solutions.

5. Validation

We validate Morph-KGC*star* by assessing 1) the engine's conformance with respect to the RML-star specification using RML-star test cases derived from the N-Triples-star syntax tests (Section 5.1); 2) its feasibility by applying it in two real-world use cases for software metadata extraction [29] (SoMEF) and biomedical research literature [30] (SemMedDB). For each use case, we evaluate a) the generation of triples with Morph-KGC*star* for different reification approaches (Section 5.2.1), and b) the time performance of Morph-KGC*star* in comparison with the SPARQL-Anything engine [12, 13] to assess our RML-based solution against a SPARQL-based solution (Section 5.2.2). To the best of our knowledge, SPARQL-Anything is the only open source knowledge graph construction engine able to generate RDF-star datasets apart from Morph-KGC*star*.

5.1. RML-star Test Cases

Test cases are commonly used to evaluate the conformance of an engine with respect to a language specification (e.g., RML test cases [31]). A set of RDF-star test cases was proposed covering the syntax of various of its serializations[11]. We adapted these test cases to evaluate the conformance of Morph-KGC*star* with respect to RML-star.

To create a representative set of test cases for RML-star, we selected the N-Triples-star syntax tests[12], given that Morph-KGC*star* generates the output RDF-star graph in this serialization. For each RDF-star test case, we created two associated RML-star test cases that generate the original RDF-star dataset: one test case with a single input data source (i.e., the mapping does not include joins) and another with two input data sources (i.e., the mapping includes joins among triple maps). For each test case, we manually created the input source(s) in the CSV format and the corresponding RML-star mapping rules to generate the output RDF-star datasets. Following this approach, we obtained 16 RML-star test cases. The test cases are openly available at the W3C Community Group on Knowledge Graph Construction [32], and can be reused by any engine to test its conformance with respect to RML-star. Morph-KGC*star* passes all test cases successfully. As stated in Section 4, all RML-star, R2RML and RML test cases were added to the continuous integration pipeline of our engine, following best practices in software development.

5.2. Use Cases

We applied Morph-KGC*star* in two real-world use cases. The first generates RDF-star graphs from scientific software documentation, and the second annotates statements extracted from biomedical research publications.

Scientific Software Metadata Extraction. Scientific software has become a crucial asset to deliver and reproduce the results described in research publications [33]. However, scientific software is often time consuming to understand and reuse due to incomplete and heterogeneous documentation, available only in a human-readable manner. The Software Metadata Extraction Framework (SoMEF) [34] proposes an approach to automatically extract relevant metadata (description, installation instructions, citation, etc.) from code repositories and their documentation. SoMEF includes different text extraction techniques (e.g., supervised classification, regular expressions, etc.) that yield results with different confidence values. For example, Listing 8 shows a JSON snippet with the description that SoMEF obtained from a software repository (Widoco) using the GitHub API. The confidence in this case is high as the extracted description was manually curated by the creators of the code repository. SoMEF extracts more than 30 different metadata fields about software, its source code, its released versions, and their corresponding authors. For transforming the output of SoMEF into RDF-star, we used a total of 35 triples maps to annotate software metadata fields and an additional triples map to annotate source code descriptions. All reified triples follow the same structure (Listings 8 & 9), i.e. the standard RDF triple contains the excerpt of the extracted feature, and it is annotated with

[11] https://w3c.github.io/rdf-star/tests/
[12] https://w3c.github.io/rdf-star/tests/nt/syntax

the *technique* used and the *confidence* value. The complete mapping and all input examples and results are available online [35].

```
"codeRepository": "https://github.com/oeg-upm/Widoco",
"description": [
   {
    "confidence": [
     1.0
    ],
    "excerpt": "Wizard for documenting ontologies. WIDOCO is ...",
    "technique": "GitHub API"
   }
 ]
```

Listing 8: JSON snippet showing the description metadata field extracted by SoMEF on a code repository using the GitHub API as extraction technique.

Capturing the technique used and the confidence obtained for each extracted metadata field is key for obtaining an accurate representation of the result. Hence, the RDF-star representation corresponding to the JSON in Listing 8 includes this information, as depicted in Listing 9.

```
ex:oeg-upm/Widoco :description "Wizard for documenting ontologies. WIDOCO is ..." .
<<ex:oeg-upm/Widoco :description "Wizard for documenting ontologies. WIDOCO is ...">>
   :technique "GitHub API" .
<<ex:oeg-upm/Widoco :description "Wizard for documenting ontologies. WIDOCO is ...">>
   :confidence "1.0" .
```

Listing 9: RDF-star triples snippet showing the results generated for the description field in Listing 8. Each asserted triple is annotated with its corresponding confidence and technique.

Biomedical Research Literature. SemMedDB [30], the Semantic MEDLINE Database, is a repository that contains information on extracted biomedical entities and predications (subject-predicate-object triples) from biomedical texts (titles and abstracts from PubMed citations). The tables comprising SemMedDB are available for download as a relational database or CSV files[13]. We downloaded the MySQL files for (1) predication predictions (PREDICATION and PREDICATION_AUX tables), containing more than 117 million annotations; and (2) entity predictions (ENTITY table), which include more than 410 million annotations. Listings 10, 11 and 12 illustrate the columns used from the tables with synthetic data. For predications, only data for subjects is shown; the missing columns regarding objects follow the same structure as subjects. Subjects and objects, from predications, and entities are assigned a semantic type (that categorizes the extracted concept in the biomedical domain) annotated with a confidence score. In addition, the extraction of subjects and objects is assigned a timestamp on when it took place. Thus, the score and timestamp represent metadata about other statements. We created an RML-star mapping with 5 triples maps quoting triples: 3 of them are used to annotate the assignation of semantic types to entities, subjects, and objects with confidence scores; the remaining 2 provide the timestamps for the extraction of subjects and objects.

```
ENTITY_ID , SEMTYPE , SCORE
12345     , orga    , 790
```

Listing 10: ENTITY table snippet.

```
PREDICATION_ID , SUBJECT_SEMTYPE , SUBJECT_NAME
13579          , Semtype         , SubjName
```

Listing 11: PREDICATION table snippet.

[13] https://lhncbc.nlm.nih.gov/ii/tools/SemRep_SemMedDB_SKR/SemMedDB_download.html

```
PREDICATION_AUX_ID, PREDICATION_ID, SUBJECT_SCORE, TIMESTAMP
67890             , 13579         , 800           , 1651740766
```

Listing 12: PREDICATION_AUX table snippet.

```
<<ex:12345 sem:semanticType "orga">> sem:score "790" .
<<ex:13579 sem:subject ex:SubjName>> sem:timestamp "1651740766" .
<<ex:SubjName sem:semanticType "Semtype">> sem:score "800" .
```

Listing 13: RDF-star triples generated from data in Listings 10, 11 and 12.

5.2.1. Comparison of Morph-KGC^{star} for different reification approaches

We compare the materialization of knowledge graphs using the reification approaches discussed in Section 3, i.e. RML-star, singleton properties and standard reification. To evaluate Morph-KGC^{star} with SoMEF, we transform all 237 repositories belonging to a single GitHub organization by applying the mapping to each organization repository in a sequential manner. For SemMedDB, we randomly selected 6 million annotations from the two types of predictions (i.e. entities and predications). All mappings used for the validation are openly available [35]. However, the data in this use case is licensed under the UMLS - Metathesaurus License Agreement[14], which does not allow for its distribution but data may be accessed by obtaining an account with the UMLS license[15].

Table 1 shows a description of the reification mapping documents and the resulting execution times obtained for both use cases. Regarding mapping complexity, RML-star and singleton properties contain the same amount of triples maps, while standard reification needs fewer triples maps. As shown in Section 3, this is due to RML-star and singleton properties requiring one triples map to generate triples, and other triples map to annotate them. Instead, in the standard reification approach, triples and their annotations are created using a single triples map. The amount of predicate object maps varies considerably among the approaches. RML-star is the approach with the lowest number of predicate object maps, and as a result, produces the fewest number of triples. Meanwhile, standard reification obtains the highest values for these metrics, as this approach requires a high number of predicate object maps to reify RDF triples. RML-star is the fastest approach when it comes to the generation of the output knowledge graph.

5.2.2. Comparison with SPARQL-Anything

We also compare our proposed implementation of RML-star with SPARQL-Anything. To our knowledge, SPARQL-Anything is the only SPARQL-based tool to generate RDF-star graphs. We adapted the RML-star test cases for SPARQL-Anything, which successfully passes all of them, i.e. the engine generates valid RDF-star graphs. To illustrate the comparison, Appendix A shows an example to create the RDF-star graph in Listing 9 from the JSON file in Listing 8 using RML-star (Listing 14) and SPARQL-Anything (Listing 15).

Table 2 shows the execution times and number of triples obtained for Morph-KGC^{star} and SPARQL-Anything for both use cases. All the experiments were performed under the same conditions for Morph-KGC^{star} and SPARQL-Anything, and the resources used are publicly available [35]. The generation times are reported as the average time of three executions running on a CPU Intel(R) Xeon(R) Silver 4216 CPU @2.10GHz with 20 cores, 128 GB RAM and a SSD SAS Read-Intensive 12 GB/s. As mapping partitioning [18] has not yet been extended for RML-star, we obtained the generation times without this optimization to fairly compare the different reification approaches.

SPARQL-Anything produces an out of memory error for SemMedDB; hence, it is not able to generate the output RDF-star graph for this use case. Scalability issues, e.g., taking long time to produce results or hitting memory limits, are well-known issue of SPARQL-Anything's performance [13] which are not addressed so far.

Regarding SoMEF, SPARQL-Anything performs faster for smaller datasets than Morph-KGC^{star} while producing more results, but it hits its limits when the size of the data grows. SPARQL-Anything generates triples with empty string literals in the object when empty string values appear in the input data as opposed to Morph-KGC^{star} which does not generate triples for empty string literals. While this causes an inconsistency on the number of generated triples, deciding whether to generate terms for empty values is an open issue and, currently, different implementa-

[14] https://www.nlm.nih.gov/research/umls/knowledge_sources/metathesaurus/release/license_agreement.html
[15] An account with the UMLS license can be requested at https://www.nlm.nih.gov/databases/umls.html.

	SemMedDB				SoMEF			
	Mapping		Generation	Number of	Mapping		Generation	Number of
	TriplesMap	POM	Time (s)	Output Triples	TriplesMap	POM	Time (s)	Output Triples
RML-star	10	10	1,796	36,067,636	78	122	1,085	15,102
Singleton	10	15	1,943	75,465,497	78	158	112	16,015
Std. Reification	9	20	4,876	127,697,142	39	199	1,201	21,268

Table 1

Results of different reification approaches per use case with Morph-KGC*star*. Generation time in seconds for the SemMedDB and SoMEF use cases, with number of generated triples and characteristics of mapping rules (number of triples maps and predicate object maps (POM)).

	SemMedDB		SoMEF (single files)		SoMEF (aggregated file)	
	Generation Time (s)	Number of Output Triples	Generation Time (s)	Number of Output Triples	Generation Time (s)	Number of Output Triples
Morph-KGC*star*	1,796	36,067,636	1,085	15,102	13,86	14,821
SPARQL-Anything	Out of memory	Out of memory	630,179	15,155	Timeout	Timeout

Table 2

Comparison of SPARQL-Anything and Morph-KGC*star* with the use cases. Generation time in seconds for the SemMedDB and scientific software metadata use cases, along with the number of generated triples. For SoMEF we consider the case of a separate file for each GitHub repository (single files) and the case of a single file with all the repositories (aggregated file).

tions handle them following an ad-hoc approach. Besides the empty strings, the two implementations generate the same number of triples. SoMEF consists of many smaller files which SPARQL-Anything can handle efficiently. However, if we aggregate all the repositories used in SoMEF in a single JSON file and we obtain a JSON array of 237 objects, Morph-KGC*star* processes this dataset in less than 14 seconds, while SPARQL-anything is not able to generate the output after 48 hours[16]. These results show that SPARQL-Anything performs better for small input data sources, but Morph-KGC*star* scales for larger volumes of data, that SPARQL-Anything is not able to process.

6. Related Work

The need for describing statements about statements led to the development of tools and languages to generate structured content from heterogeneous data sources. For example, the community around large knowledge graphs, such as Wikidata [36], developed community-driven tools for qualifying statements[17] (i.e. adding qualifiers to annotate a triple). Another approach is RDF-star [1], which has been gaining popularity and adoption by the community (e.g., it has been implemented by GraphDB, Apache Jena, Stardog, etc.) as a mean to represent reified triples.

Mapping languages establish the relationships between data sources and a target ontology to create or access RDF data. The use of mapping languages to generate knowledge graphs has increased in recent years [23, 37, 38]. The W3C's R2RML [7] focuses on transformations from relational databases to RDF. Extensions of this language are developed to overcome its limitations and broaden its capabilities [38]. Among these languages, we highlight RML [8], which extends R2RML to heterogeneous data sources (e.g., CSV, JSON, etc.). Unlike R2RML-based mapping languages, which follow a custom syntax, existing languages were also repurposed to generate RDF [38]. For instance, SPARQL-Generate [11] and SPARQL-Anything extend the query language SPARQL [10], whereas ShExML extends the constraints language ShEx [14].

So far, two declarative mapping languages have been proposed to generate RDF-star graphs from heterogeneous data sources based on R2RML. RML-star [17] extends RML for which this paper contributes Morph-KGC*star* as an implementation. The other is R2RML-star [39], an extension over R2RML, for which an algorithm to trans-

[16]Note that the number of triples is different w.r.t. to single files because duplicated triples generated from different repositories are removed.
[17]https://www.wikidata.org/wiki/Help:QuickStatements

late SPARQL-star into SQL queries is provided. Unfortunately, the R2RML-star implementation is not publicly available, and, at the time of writing, the permanent URL for the R2RML-star's ontology[18] does not resolve.

SPARQL-Anything is also able to create RDF-star graphs without any extensions just by using the CONSTRUCT clause in SPARQL-star and Apache Jena. Since the implementation for R2RML-star is not openly available, its comparison with the rest of the languages and associated tools is based on its description [39]. The three proposals for RDF-star generation differ with respect to supported data, backward compatibility, and limitations:

(1) RML-star and SPARQL-Anything allow generating RDF-star from multiple heterogeneous data sources, while R2RML-star builds upon R2RML, generating RDF-star only from data in relational databases.

(2) RML-star extends RML adhering to the RML specification and remaining backward compatible: a valid RML mapping document is also a valid RML-star document. Since SPARQL-Anything is based on SPARQL-star, it also remains backward compatible. However, R2RML-star introduces changes to the R2RML ontology, which are inconsistent with the original ontology. For instance, a `rr:SubjectMap` expects a template-, column-, or constant-valued `rr:TermMap` as its range. The R2RML-star extension introduces the `star:RDFStarTermType`, a new term map type (next to `rr:IRI`, `rr:Literal` and `rr:BlankNode`), and three properties: `star:subject`, `star:predicate` and `star:object`. The range of `star:subject` and `star:object` is `rr:ObjectMap`; and `rr:PredicateMap` is the range of `star:predicate`. In this way, recursion can be achieved, since a `rr:ObjectMap` from a `star:RDFStarTermType` can be, in turn, another `star:RDFStarTermType`. However, these properties have as domain `rr:TermMap`, superclass of `rr:SubjectMap`, `rr:PredicateMap` and `rr:ObjectMap`, which allows any of these terms to have nested triples. According to the RDF-star specification, this is correct for objects and subjects, but not for predicates.

(3) RML-star and SPARQL-Anything supports joins and recursion. The R2RML-star extension enables recursion, but joins can only be performed with R2RML views. This occurs because the ranges of `star:subject` and `star:object` are `rr:ObjectMap`[19] but `rr:RefObjectMap` is not foreseen, which is the one that allows joining with other data sources.

(4) RML-star introduces a unique construct to define the quoted triples and "flags" if a quoted triple should be asserted. In R2RML-star only quoted triples are generated. If the corresponding asserted triple needs to be generated, an additional `rr:TriplesMap` needs to be defined to assert the quoted triple. Similarly, to assert a quoted triple in SPARQL-Anything, an additional triple has to be specified in the query.

RML-star, R2RML-star and SPARQL-Anything are accompanied by implementations. RML-star is implemented in this work (Morph-KGC^{star}), R2RML-star is implemented as an extension of Ontop [40] for virtual RDF-star graphs [39], while the implementation of SPARQL-Anything carries the same name as the syntax. RML-star and SPARQL-Anything follow a materialization approach, while R2RML-star follows a virtualization approach.

7. Conclusions and Future Work

In this paper, we describe Morph-KGC^{star}, an engine that generates RDF-star graphs from heterogeneous sources using the RML-star mapping language. We presented the algorithm behind the implementation and show that it produces valid RDF-star triples by creating RML-star test cases derived from the N-Triples-star syntax tests. We have also applied Morph-KGC^{star} in two real-world use cases from the biomedical and open science domains, showing that generating RDF-star data with our engine is faster than other reification alternatives. Finally, we compare our approach with SPARQL-Anything with the test cases and use cases presented, showing that Morph-KGC^{star} outperforms SPARQL-Anything processing large-sized data, but it is slower for small-sized data.

Morph-KGC^{star} is, to the best of our knowledge, the first open source engine for generating RDF-star knowledge graphs with declarative mapping rules. Given the increasing adoption of RDF-star by the Semantic Web community (e.g., graph stores, libraries or the W3C Draft Charter for an RDF-star Working Group) and the lack of tools to generate RDF-star graphs, we expect that Morph-KGC^{star} will further contribute to the adoption of RDF-star. Morph-KGC^{star} is actively maintained and will adapt to future modifications (if any) in the RDF-star specification.

[18] https://w3id.org/obda/r2rmlstar#

[19] In fact, if the `star:subject` is an `rr:ObjectMap`, it allows generating literals as subjects, which is not valid RDF.

Our future work includes adding new features to Morph-KGC^{star}, such as supporting NoSQL databases and simpler, human-readable mappings (extending YARRRML [41] to RML-star). We also plan to improve the performance of RDF-star materialization, e.g., by extending mapping partitioning [18] to RML-star.

Acknowledgements

This work was partially funded by the project "Knowledge Spaces: Técnicas y herramientas para la gestión de grafos de conocimientos para dar soporte a espacios de datos" (Grant PID2020-118274RB-I00, funded by MCIN/AEI/ 10.13039/501100011033) and by the Euratom Research and Training Programme 2019-2020 ENTENTE under Grant 900018. David Chaves-Fraga is supported by Ministerio de Universidades, Spain and by the NextGenerationEU funds through the Margarita Salas postdoctoral fellowship. Daniel Garijo is supported by Comunidad de Madrid, Spain under the Multiannual Agreement with Universidad Politécnica de Madrid in the line Support for R&D projects for Beatriz Galindo researchers. Anastasia Dimou and David Chaves-Fraga are also supported by Flanders Make.

References

[1] O. Hartig, 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. http://ceur-ws.org/Vol-1912/paper12.pdf.

[2] R. Cyganiak, D. Wood and M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, World Wide Web Consortium (W3C), 2014. https://www.w3.org/TR/rdf11-concepts/.

[3] D. Hernández, A. Hogan and M. Krötzsch, Reifying RDF: What Works Well With Wikidata?, in: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems*, CEUR Workshop Proceedings, Vol. 1457, 2015, pp. 32–47. http://ceur-ws.org/Vol-1457/SSWS2015_paper3.pdf.

[4] P.J. Hayes and P.F. Patel-Schneider, RDF 1.1 Semantics, W3C Recommendation, World Wide Web Consortium (W3C), 2014. http://www.w3.org/TR/rdf11-mt/.

[5] V. Nguyen, O. Bodenreider and A. Sheth, Don't like RDF Reification? Making Statements about Statements Using Singleton Property, in: *Proceedings of the 23rd International Conference on World Wide Web*, Association for Computing Machinery, 2014, pp. 759—770. ISBN 9781450327442. doi:10.1145/2566486.2567973.

[6] O. Hartig, P.-A. Champin, G. Kellogg and A. Seaborne, RDF-star and SPARQL-star, W3C Final Community Group Report, 2021. https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html.

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

[8] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: *Proceedings of the 7th Workshop on Linked Data on the Web*, CEUR Workshop Proceedings, Vol. 1184, 2014. ISSN 1613-0073. http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[9] F. Michel, L. Djimenou, C.F. Zucker and J. Montagnat, Translation of Relational and Non-Relational Databases into RDF with xR2RML, in: *Proceedings of the 11th International Conference on Web Information Systems and Technologies*, Vol. 1, SciTePress, 2015, pp. 443–454. ISSN 2184-3252. ISBN 978-989-758-106-9. doi:10.5220/0005448304430454.

[10] E. Prud'hommeaux and A. Seaborne, SPARQL Query Language for RDF, Recommendation, World Wide Web Consortium (W3C), 2008. http://www.w3.org/TR/rdf-sparql-query/.

[11] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL Extension for Generating RDF from Heterogeneous Formats, in: *Proceedings of the 14th Extended Semantic Web Conference*, Springer International Publishing, 2017, pp. 35–50. ISBN 978-3-319-58068-5.

[12] E. Daga, L. Asprino, P. Mulholland and A. Gangemi, Facade-X: an opinionated approach to SPARQL anything, *Studies on the Semantic Web* **53** (2021), 58–73.

[13] L. Asprino, E. Daga, A. Gangemi and P. Mulholland, Knowledge Graph Construction with a Façade: a Unified Method to Access Heterogeneous Data Sources on the Web, *Transactions on Internet Technology* (2022), accepted for publication.

[14] E. Prud'hommeaux, J.E. Labra Gayo and H. Solbrig, Shape Expressions: An RDF Validation and Transformation Language, in: *Proceedings of the 10th International Conference on Semantic Systems*, Association for Computing Machinery, 2014, pp. 32–40–. ISBN 9781450329279. doi:10.1145/2660517.2660523.

[15] H. García-González, I. Boneva, S. Staworko, J.E. Labra-Gayo and J.M.C. Lovelle, ShExML: improving the usability of heterogeneous data mapping languages for first-time users, *PeerJ Computer Science* **6** (2020), e318. doi:https://doi.org/10.7717/peerj-cs.318.

[16] Apache Software Foundation, Apache Jena, 2021. https://jena.apache.org.

[17] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga and A. Dimou, RML-star: A Declarative Mapping Language for RDF-star Generation, in: *International Semantic Web Conference, ISWC, P&D*, CEUR Workshop Proceedings, Vol. 2980, CEUR-WS.org, 2021. http://ceur-ws.org/Vol-2980/paper374.pdf.
[18] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M.S. Pérez and O. Corcho, Morph-KGC: Scalable Knowledge Graph Materialization with Mapping Partitions, *Semantic Web Journal* (2022). http://www.semantic-web-journal.net/system/files/swj3135.pdf.
[19] F. Manola and E. Miller, RDF primer, W3C Recommendation, World Wide Web Consortium (W3C), 2004. https://www.w3.org/TR/rdf-primer/.
[20] R. Dividino, S. Sizov, S. Staab and B. Schueler, Querying for provenance, trust, uncertainty and other meta knowledge in RDF, *Journal of Web Semantics* **7**(3) (2009), 204–219. doi:https://doi.org/10.1016/j.websem.2009.07.004.
[21] S.C. Feria, R. García-Castro and M. Poveda-Villalón, Chowlk: from UML-Based Ontology Conceptualizations to OWL, in: *Proceedings of the 19th Extended Semantic Web Conference*, Springer International Publishing, 2022, pp. 338–352. ISBN 978-3-031-06981-9.
[22] A. Iglesias-Molina, J. Arenas-Guerrero, T. Delva, A. Dimou and D. Chaves-Fraga, RML-star, W3C Draft Community Group Report, 2022. https://kg-construct.github.io/rml-star-spec/.
[23] J. Arenas-Guerrero, M. Scrocca, A. Iglesias-Molina, J. Toledo, L. Pozo-Gilo, D. Doña, O. Corcho and D. Chaves-Fraga, Knowledge Graph Construction with R2RML and RML: An ETL System-based Overview, in: *Proceedings of the 2nd International Workshop on Knowledge Graph Construction*, CEUR Workshop Proceedings, Vol. 2873, 2021. http://ceur-ws.org/Vol-2873/paper11.pdf.
[24] W. McKinney, Data Structures for Statistical Computing in Python, in: *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56–61. doi:10.25080/Majora-92bf1922-00a.
[25] Bayer, Michael, SQLAlchemy, in: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*, aosabook.org, 2012. http://aosabook.org/en/sqlalchemy.html.
[26] G.A. Grimnes, N. Car, G. Higgins, J. Hees, I. Aucamp, N. Arndt, A. Sommer, E. Chuc, I. Herman, A. Nelson, N. Lindström, T. Gillespie, T. Kluyver, F. Ludwig, P.-A. Champin, M. Watts, U. Holzer, D. Winston, R. Chateauneu, B. Cogrel, W. Haruna, D. Krech, C. Markiewicz, JervenBolleman, D. Scott, D. Perttula and J. McCusker, RDFLib/rdflib: RDFlib 6.2.0, Zenodo, 2022. doi:10.5281/zenodo.6845246.
[27] A. Makinouchi, A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model, in: *Proceedings of the 3rd International Conference on Very Large Data Bases*, VLDB Endowment, 1977, pp. 447–453–.
[28] J.A. Guerrero, J. Toledo and D. Chaves, oeg-upm/morph-kgc: 2.0.0, Zenodo, 2022. doi:10.5281/zenodo.6472343.
[29] A. Kelley and D. Garijo, A Framework for Creating Knowledge Graphs of Scientific Software Metadata, *Quantitative Science Studies* (2021), 1–37. https://doi.org/10.1162/qss_a_00167.
[30] H. Kilicoglu, D. Shin, M. Fiszman, G. Rosemblat and T.C. Rindflesch, SemMedDB: a PubMed-scale repository of biomedical semantic predications, *Bioinformatics* **28**(23) (2012), 3158–3160. doi:10.1093/bioinformatics/bts591.
[31] P. Heyvaert, D. Chaves-Fraga, F. Priyatna, O. Corcho, E. Mannens, R. Verborgh and A. Dimou, Conformance test cases for the RDF mapping language (RML), in: *Proceedings of the 1st Iberoamerican Knowledge Graphs and Semantic Web Conference*, Springer, 2019, pp. 162–173.
[32] D. Chaves, A. Iglesias, D. Garijo and J.A. Guerrero, kg-construct/rml-star-test-cases: v1.1, Zenodo, 2022. doi:10.5281/zenodo.6518802.
[33] N.P. Chue Hong, D.S. Katz, M. Barker, A.-L. Lamprecht, C. Martinez, F.E. Psomopoulos, J. Harrow, L.J. Castro, M. Gruenpeter, P.A. Martinez and T. Honeyman, FAIR Principles for Research Software (FAIR4RS Principles) (2021). doi:10.15497/RDA00068.
[34] A. Mao, D. Garijo and S. Fakhraei, SoMEF: A Framework for Capturing Scientific Software Metadata from its Documentation, in: *2019 IEEE International Conference on Big Data*, 2019, pp. 3032–3037. doi:10.1109/BigData47090.2019.9006447.
[35] D. Chaves, A. Iglesias and D. Garijo, oeg-upm/rdf-star-generation: v1.0, Zenodo, 2022. doi:10.5281/zenodo.6919707.
[36] D. Vrandečić and M. Krötzsch, Wikidata: A Free Collaborative Knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85. doi:10.1145/2629489.
[37] G. Xiao, L. Ding, B. Cogrel and D. Calvanese, Virtual Knowledge Graphs: An Overview of Systems and Use Cases, *Data Intelligence* **1**(3) (2019), 201–223. doi:10.1162/dint_a_00011.
[38] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester and A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data, *Journal of Web Semantics* (2022), accepted for publication.
[39] L. Sundqvist, Extending VKG Systems with RDF-star Support, 2022. https://ontop-vkg.org/publications/2022-sundqvist-rdf-star-ontop-msc-thesis.pdf.
[40] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalaycı, L. Ding, J. Corman, B. Cogrel, D. Calvanese and E. Botoeva, The Virtual Knowledge Graph System Ontop, in: *Proceedings of the 19th International Semantic Web Conference, ISWC*, Springer International Publishing, 2020, pp. 259–277. ISBN 978-3-030-62466-8. doi:10.1007/978-3-030-62466-8_17.
[41] P. Heyvaert, B. De Meester, A. Dimou and R. Verborgh, Declarative Rules for Linked Data Generation at Your Fingertips!, in: *Extended Semantic Web Conference*, Springer International Publishing, 2018, pp. 213–217. ISBN 978-3-319-98192-5. doi:10.1007/978-3-319-98192-5_40.
[42] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487. doi:10.3233/SW-160217.
[43] C. Debruyne and D. O'Sullivan, R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings, in: *Proceedings of the 9th Workshop on Linked Data on the Web*, 2016. http://ceur-ws.org/Vol-1593/article-13.pdf.

Appendix A. SPARQL-Anything and RML-star mappings

```
:ls a rml:LogicalSource ;
  rml:source "./data/somef/morph.json";
  rml:referenceFormulation ql:JSONPath ; rml:iterator "$" .

:soft a rr:SubjectMap ;
   rr:template "https://www.w3id.org/okn/i/Software/{owner.excerpt}/{name.excerpt}" ;
   rr:class sd:Software.

:descriptionTM rml:logicalSource :ls;
    rml:subjectMap :soft;
    rr:predicateObjectMap [
       rr:predicate sd:description ;
       rml:objectMap [
          rml:reference "description.excerpt" ;
          rr:termType rr:Literal ] ].

:descriptionMetadataTM rml:logicalSource :ls ;
  rml:subjectMap [ rml:quotedTriplesMap :descriptionTM ] ;
  rr:predicateObjectMap [
    rr:predicate em:confidence ;
    rml:objectMap [ rml:reference "description.confidence"] ] ;
  rr:predicateObjectMap [
    rr:predicate em:technique ;
    rml:objectMap [ rml:reference "description.technique"] ] .
```

Listing 14: RML-star mapping to create the RDF-star graph in Listing 9 from the JSON file in Listing 8.

```
CONSTRUCT {
    ?subject a sd:Software ;
       sd:description ?desc_excerpt .
    << ?subject sd:description ?desc_excerpt >> em:technique ?desc_technique ;
       em:confidence ?desc_confidence . }
WHERE
  { SERVICE <x-sparql-anything:./data/somef/morph.json,json.path=$.owner>
      { [] xyz:excerpt ?owner ;
           xyz:confidence [ fx:anySlot ?owner_confidence ] ;
           xyz:technique ?owner_technique . }
    SERVICE <x-sparql-anything:./data/somef/morph.json,json.path=$.name>
      { [] xyz:excerpt ?name . }

  BIND(uri(concat("https://www.w3id.org/okn/i/Agent/",?owner)) as ?owner_uri)
  BIND(uri(concat(:i,encode_for_uri(?owner),"/",encode_for_uri(?name))) as ?subject)

  OPTIONAL
    { SERVICE <x-sparql-anything:./data/somef/morph.json,json.path=$.description>
       { [] xyz:excerpt ?desc_excerpt ;
            xyz:confidence [ fx:anySlot ?desc_confidence ] ;
            xyz:technique ?desc_technique . } }
```

Listing 15: SPARQL-Anything snippet to create the RDF-star graph in Listing 9 from the JSON file in Listing 8.