Semantic Web -1 (2024) 1–19
DOI 10.3233/SW-243602
IOS Press

Declarative generation of RDF-star graphs from heterogeneous data

Julián Arenas-Guerrero [a,*,**], Ana Iglesias-Molina [a,**], David Chaves-Fraga [d,b,c,a], 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-mail: anastasia.dimou@kuleuven.be
[c] *Flanders Make, DTAI-FET, Belgium*
[d] *Grupo de Sistemas Intelixentes, Universidade de Santiago de Compostela, Spain*
E-mail: david.chaves@usc.es

Editors: Mehwish Alam, Télécom Paris, Institut Polytechnique de Paris, France; Ruben Verborgh, Ghent University – imec, Belgium
Solicited reviews: Sebastián Ferrada, Universidad de Chile, Chile; Pierre-Antoine Champin, LIRIS, France; Kai Eckert, Mannheim University of Applied Sciences, Germany

Abstract. RDF-star has been proposed as an extension of RDF to make statements about statements. 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 system 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 (originally, *RDF** [19]) was proposed as an extension of RDF [11] to make statements about other statements (also known as reification [23]). RDF-star extends RDF's conceptual data model and concrete syntaxes by providing a compact alternative to other reification approaches, such as *standard reification* [22] or *singleton properties* [37]. Following the uptake of the initial version of RDF-star, the W3C RDF-DEV Community Group[1] released a W3C Final Community Group Report [21] and the RDF-star Working Group[2] was formed to extend related W3C Recommendations.

[*] 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

Although 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 is either custom or repurposed. The syntax of custom mapping languages is designed to generate RDF graphs, such as the W3C Recommendation R2RML [13], for generating RDF from data in relational databases, and its extensions for heterogeneous data, e.g., RDF Mapping Language (RML) [28] or xR2RML [36]. Alternatively, mapping languages may repurpose an existing syntax proposed for other scopes, e.g., based on the query language SPARQL [40], such as SPARQL-Generate [32] or SPARQL-Anything [6,12], or on the constraints language ShEx [39], such as ShExML [17].

Mapping languages have focused so far on the generation of RDF graphs, but the emergence of RDF-star presents 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 [21] as long as their adjustments to the syntax are not affected and the implementation on which they are based allows it. For instance, SPARQL-Anything is built on top of Apache Jena [1], which supports 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 [14], 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 [4]; (iv) a validation of the algorithm and its implementation based on test and use cases; (v) a comparison of our proposal against other approaches to generate reified RDF (standard reification and singleton properties) in terms of generation time; and (vi) a comparison with SPARQL-Anything, a SPARQL-based language to generate RDF-star graphs.

The remainder 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 article 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 [21] was proposed as an extension of RDF to concisely make statements about statements, represented as RDF triples. RDF-star captures the notion of "*quoted triple*", which are enclosed in the concrete syntaxes using "«" and "»". An RDF-star triple can be placed in the subject or object of another RDF-star triple. 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 [28] extends the W3C Recommendation R2RML [13] 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

[3] https://w3c.github.io/rdf-star/implementations

document is a set of `rml:TriplesMap`, each of them containing one `rml:LogicalSource`, one `rml:SubjectMap`, and from zero to multiple `rml:PredicateObjectMap`. The `rml:SubjectMap` declares how the subject of the triples are generated and also indicates its class, using the property `rml:class`. A `rml:PredicateObjectMap` contains one or more `rml:PredicateMap` to define the predicates of the triples and, in a similar way, one or more `rml:ObjectMap` that declare how the objects should be generated. Subject maps and predicate-object maps can have from zero to multiple `rml:GraphMap`, which describe how to generate named graphs (if generated). When a join between logical sources is needed, `rml:ObjectMap` is replaced by `rml:RefObjectMap`, which uses the subject map of a triples map (`rml: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 `rml:joinCondition`, `rml:child` and `rml:parent`. Subject, predicate, object and graph maps are `rml: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 generated from a data field), or template (the RDF terms are obtained 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 first W3C Recommendation of RDF [31] already included a description of the standard reification approach. Other alternatives were proposed over the years, such as singleton properties [37], RDF$^+$ [15], and more recently, RDF-star [19]. This section describes some reification approaches and shows how they can be used in RML and RML-star.

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`) and its score (`SCORE`), 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".

3.1. *Reification with RML*

Two reification approaches stand out: standard reification and singleton properties. These approaches use strategies that add metadata to triples without additional constructs (as opposed to e.g., named graphs [23]). 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 [31] was proposed in the first W3C Recommendation of RDF. 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`. In 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 statements with `:date`.

Singleton properties [37] 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`.

```
ID , DATE       , MARK , PERSON   , SCORE
1  , 2022-03-21 , 4.80 , Angelica , 1211
2  , 2022-03-19 , 4.85 , Katerina , 1224
```

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

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

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

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

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

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

3.2. *Reification with RML-star*

In a previous work [14], we proposed RML-star (Fig. 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, a quoted triple may appear only in the subject or object of a triple. Thus, 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. The referenced triples map can only be one of the following: (i) `rml:AssertedTriplesMap` to be asserted in the output graph, or (ii) `rml:NonAssertedTriplesMap`

```
_: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

```
: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

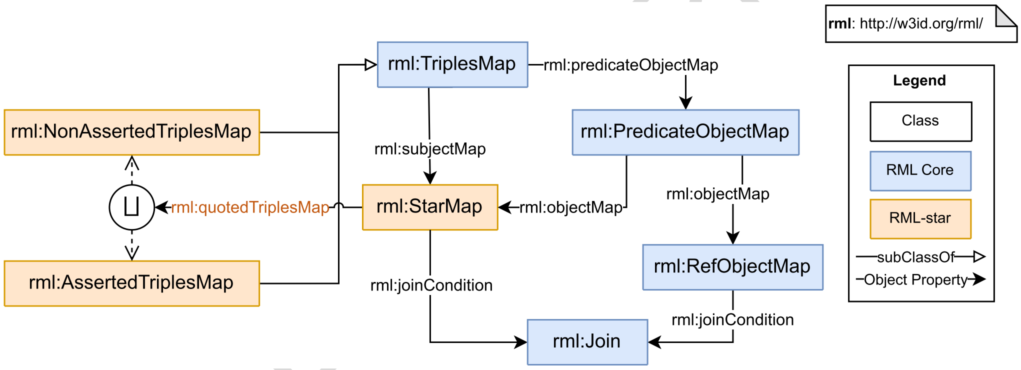

Fig. 1. The RML-star module (represented using the Chowlk Visual Notation [16]). The RML-star resources are highlighted in orange, while the rest of the represented ontology belongs to the RML-Core module.[4].

to not be asserted. A quoted triples map can specify `rml:class` in the subject map or have multiple predicate-object maps. Thus, several triples sharing the same subject map and logical source can be quoted by the same star map. An example of an RML-star mapping rule is shown in Listing 6, which generates the RDF-star triples in Listing 7 from data in Listing 1. The mapping rules use an asserted triples map (`<#jumpTM>`) within the subject map of a triples map (`<#dateTM>`). The quoted triples map `<#jumpTM>` contains two predicate-object maps that produce triples that are annotated with `:date` by the triples map `<#dateTM>`. The first predicate-object map (*lines 6-9*) produces the triples for the height of the jump, the same as the examples presented previously for standard reification in Listing 2 and singleton properties in Listing 3. We extend the RML-star example with a second predicate-object map to also represent the score of the jump within the same quoted triples map (*lines 10-13*). To produce this triple in the other reification approaches an additional triples map for each case would be required.

Currently, the RML-star specification [26] provides a complete description of the language, is published as a W3C Draft Community Group Report, and is maintained by the W3C Knowledge Graph Construction Community

[4] http://w3id.org/rml/core/spec

```
<#jumpTM>
  a rml:AssertedTriplesMap ;
  rml:logicalSource :marks ;
  rml:subjectMap [
    rml:template ":{PERSON}" ] ;
  rml:predicateObjectMap [
    rml:predicate :jumps ;
    rml:objectMap [
      rml:reference "MARK" ] ] ;
  rml:predicateObjectMap [
    rml:predicate :score ;
    rml:objectMap [
      rml:reference "SCORE" ] ] .
```

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

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

```
:Angelica :jumps "4.80" .
<< :Angelica :jumps "4.80" >> :date "2022-03-21" .
:Katerina :jumps "4.85" .
<< :Katerina :jumps "4.85" >> :date "2022-03-19" .
:Angelica :score "1211" .
<< :Angelica :score "1211" >> :date "2022-03-21" .
:Katerina :score "1224" .
<< :Katerina :score "1224" >> :date "2022-03-19" .
```

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

Group.[5] This extension belongs to the modules that comprise the new RML specification[6] that is currently under development by the aforementioned Community Group. The semantics of the relationships between the old and this new specification are defined and available online[7] to facilitate backwards compatibility. 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`.

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.

Here, we assume that the mappings are normalized, as defined by Rodríguez-Muro & Rezk [41]. All triples maps in a normalized mapping document contain a subject map, a single predicate-object map with one predicate map and one object map and, optionally, one graph map. Any mapping document can be normalized, and we refer to a normalized triples map as a *mapping rule*. It must be noted that Morph-KGC^{star} applies mapping normalization as a preprocessing step, and it then applies Algorithm 1, further described in this section.

4.1. Materialization with RML-star

The materialization of an RML-star mapping rule is presented in Algorithm 1. It takes the following parameters as input: (i) the mapping rule to be processed (m); (ii) the complete set of mapping rules in the mapping document

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

[6] http://w3id.org/rml/portal/

[7] https://w3id.org/rml/portal/backwards-compatibility

Algorithm 1 Materialization of an RML-star rule

```
Require: $m$: mapping rule
Require: $M$: set of rules in the mapping document
Require: $nestLevel$: depth of recursion of the function call
                              ▷ $SM$, $PM$, $OM$ and $GM$ refer to subject, predicate, object and graph map
procedure MATERIALIZEMAPPINGRULE($m$, $M$, $nestLevel = 0$)
    if ISNONASSERTED($m$) and $nestLevel == 0$ then
        return
    end if
    if ISSIMPLETERMMAP($m.SM$) then
        $subjects \leftarrow$ MATERIALIZETERMMAP($m.SM$)
    else if ISSTARTERMMAP($m.SM$) then
        $m_{parent} \leftarrow$ GETMAPPINGRULE($m.SM$, $M$)
        $m_{joint} \leftarrow$ JOINMAPPINGRULES($m$, $m_{parent}$)
        $subjects \leftarrow$ MATERIALIZEMAPPINGRULE($m_{joint}$, $M$, $nestLevel + 1$)
    end if
    if ISSIMPLETERMMAP($m.OM$) then
        $objects \leftarrow$ MATERIALIZETERMMAP($m.OM$)
    else if ISREFTERMMAP($m.OM$) then
        $objects \leftarrow$ MATERIALIZEREFTERMMAP($m.OM$, $M$)
    else if ISSTARTERMMAP($m.OM$) then
        $m_{parent} \leftarrow$ GETMAPPINGRULE($m.OM$, $M$)
        $m_{joint} \leftarrow$ JOINMAPPINGRULES($m$, $m_{parent}$)
        $objects \leftarrow$ MATERIALIZEMAPPINGRULE($m_{joint}$, $M$, $nestLevel + 1$)
    end if
    $predicates \leftarrow$ MATERIALIZETERMMAP($m.PM$)
    if $nestLevel == 0$ then
        if HASGRAPHMAP($m$) then
            $namedGraphs \leftarrow$ MATERIALIZETERMMAP($m.GM$)
            return CREATEQUADS($subjects$, $predicates$, $objects$, $namedGraphs$)
        else
            return CREATETRIPLES($subjects$, $predicates$, $objects$)
        end if
    else if $nestLevel > 0$ then
        return CREATESTARTRIPLES($subjects$, $predicates$, $objects$)
    end if
end procedure
```

(M), needed to retrieve nested rules; and (iii) the nesting level of a rule ($nestLevel$), with 0 referring to the lack of nesting. 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 are obtained by iterating over its predicate-object, predicate, object, and graph maps, so that only one subject, predicate, object, and graph map are processed at a time. Note that the R2RML Recommendation[8] endorses processing triples maps in this way.

Triples maps The principal point to consider when processing an RML-star rule is that it 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. The general idea of Algorithm 1 is traversing 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

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

```
<#jumpTM>
  a rml:NonAssertedTriplesMap ;
  rml:logicalSource :marks ;
  rml:subjectMap [
    rml:template ":{PERSON}" ] ;
  rml:predicateObjectMap [
    rml:predicate :jumps ;
    rml:objectMap [
      rml:reference "MARK" ] ] ;
<#dateTM>
  a rml:TriplesMap ;
  rml:logicalSource :marks ;
  rml:subjectMap [
    rml:quotedTriplesMap <#jumpTM> ] ;
  rml:predicateObjectMap [
    rml:predicate :date ;
    rml:objectMap [
      rml:reference "DATE" ] ] .
```

Listing 8. Running mapping example to describe the RML-star materialization procedure. The mapping transforms data in Listing 1, considering one triples map with a star map referring to a non-asserted triples map

map in the object), and finally the current mapping rule is processed for generating the quoted triples. Hereinafter, we refer to the mapping rule at 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. In the following, Algorithm 1 is explained in detail together with a running example using the mapping in Listing 8, which contains the <#dateTM> triples map with a star map in the subject referring to the <#jumpTM> non-asserted triples map. This example uses a basic mapping, simplified from the one presented in previous sections so as to facilitate the understanding of the algorithm.

Non-asserted triples maps First, **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 map must only be processed when generating quoted triples. Algorithm 1 uses the nestLevel parameter to keep track of the level of nesting that 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 immediately discarded by Algorithm 1 (*lines 2-3*) as the triples that it generates should not be asserted. If nestLevel is not 0, the generated triples will be quoted and the mapping rule should be processed.

Running Example 1.1. To generate the output RDF-star graph, the algorithm is executed twice, once for each triples map in the input mapping document. First, when the algorithm is applied to the <#jumpTM> triples map, the rule is not executed (*lines 2-3*), as the triples map is non-asserted and should only generate quoted triples. Then, the algorithm is applied to the <#dateTM> triples map which is processed as it is asserted. In this execution, the <#jumpTM> triples map will be later processed, as it is a quoted triples map within the star map in the <#dateTM> triples map.

Term maps Next, **term maps** are processed to generate the terms of the triples. There are three types of maps in RML-star that need to be differentiated for materialization: term maps, referencing object maps, and star maps. When evaluating a term map, Algorithm 1 checks its type (for instance, for object maps it is checked in *lines 12, 14 & 16*, with *isSimpleTermMap(m.OM)* evaluating to true if the object map of a rule is a simple term map) and then processes the term map according to it. The handling of simple and referencing term maps (covered in *lines 6, 13, 15 & 21*) is already considered in R2RML and RML materialization procedures that are well reported in the literature [5] and more details of their materialization can be found in the R2RML Recommendation. When the type of a map resolves to a star map (*lines 7 & 16*), it is managed as described next.

Running Example 1.2. The <#dateTM> rule has two simple term maps in the predicate (rml:constant :date) and object (rml:reference "DATE") positions and one star map in the subject position; while #jumpTM contains only simple term maps. The simple term maps in <#dateTM> are evaluated after the star map, and the term maps in #jumpTM are evaluated in the recursive call of the procedure when evaluating the star map in <#dateTM>.

Star maps **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 mapping rule is retrieved from the set of mapping rules *M* (*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 and any of the original logical sources (child or parent) can be used as the joint logical source. As star maps entail 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. 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.

Running Example 1.3. In `<#dateTM>`, the subject map was previously evaluated as a star map in *line 7*. The mapping rule given by `<#jumpTM>` is retrieved as the parent rule in *line 8*. The join between the rules (*line 9*) has no effect as both of them have the same logical source, and the logical source of the parent is used as the joint logical source. Last, the quoted subject terms are generated in a recursive call of the procedure passing the joint mapping rule as an argument (*line 10*).

Combining subject, predicate and object terms to generate the output triples is done in a similar way than in standard RML (without the RML-star extension). This combination is done row-wise, one triple is generated for each row in the (joint) logical source of the mapping rule.

Graph maps Finally, the generation of **quads** must be considered. In RDF-star, quads are never quoted. However, in RML-star, triples maps are not restricted from having a graph map (i.e., inner mapping rules 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 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 does not have a graph map, RDF-star triples are added to the default graph of the output dataset (*lines 26-27*). When processing an inner mapping rule, the generated triples must be quoted (*lines 29-30*), i.e., enclosed with "«" and "»".

Running Example 1.4. In the case of `<#dateTM>`, the level of nesting is 0 (*line 22*) and it has no graph map (*line 23*), so plain triples are created (*line 27*). When processing `<#jumpTM>` as a result of the aforementioned recursive call, the level of nesting is 1 (*line 29*), and star triples will be created (*line 30*) enclosed by "«" and "»".

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

Morph-KGC [4] is an R2RML and RML compliant materialization engine implemented in Python and using Pandas [35] 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 [8] to access relational databases. In this way, many popular database management systems are supported. In addition, it allows for a wide range of tabular data sources powered by Pandas (CSV, Apache Parquet, Apache 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}.

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 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 [38] store populated with RDF-star triples. We integrated Morph-KGC^{star} with Oxigraph, as Morph-KGC only integrated originally with RDFLib [18], 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.

We ensure backward compatibility with R2RML and the legacy RML specification as follows. If a set of mapping rules is provided to Morph-KGC^{star} in the legacy RML or R2RML languages, it will be translated to the new version of RML (specifically, using the RML-Core and RML-star modules). This step is performed by replacing the R2RML and legacy RML terms with those provided by the new RML vocabulary (e.g., `rr:subjectMap` is converted to `rml:subjectMap`). 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.[9]

[9] https://www.w3.org/TR/r2rml/#natural-mapping

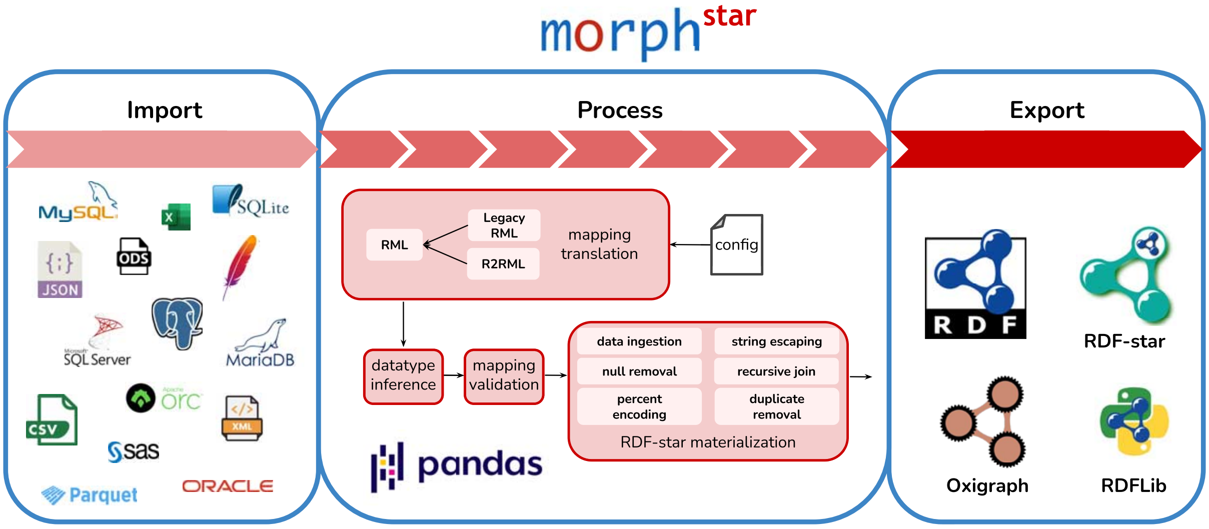

Fig. 2. Overview of Morph-KGC^{star}.

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 [33], 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[10] and the engine is distributed as a PyPi package.[11] 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 archived at Zenodo [2]. 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.[12]

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 [5] 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 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 (i) 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); (ii) 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 performance of Morph-KGC^{star} and SPARQL-Anything [6,12] to compare our RML-based solution against a SPARQL-based solution (Section 5.2.2). To the best of our knowledge,

[10] https://github.com/morph-kgc/morph-kgc
[11] https://pypi.org/project/morph-kgc/
[12] https://morph-kgc.readthedocs.io

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 [24]). A set of RDF-star test cases was proposed covering the syntax of various of its serializations.[13] 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,[14] 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 [10], 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 mentioned 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 for delivering and reproducing the results described in research publications [7]. 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 9 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. To transform the output of SoMEF into RDF-star, we used a total of 35 triples maps to annotate software metadata

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

Listing 9. JSON snippet showing the description metadata field extracted by SoMEF on a code repository using the GitHub API as extraction technique

[13] https://w3c.github.io/rdf-star/tests/

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

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

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

Listing 11. ENTITY table snippet

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

Listing 12. PREDICATION table snippet

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

Listing 13. 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 14. RDF-star triples generated from data in Listings 11, 12 and 13

fields and an additional triples map to annotate source code descriptions. All reified triples follow the same structure (Listings 9 & 10), i.e., the standard RDF triple contains the excerpt of the extracted feature, and it is annotated with the *technique* used and the *confidence* value. The complete mapping and all input examples and results are available online [9].

Capturing the technique used and the confidence obtained for each extracted metadata field is key to obtain an accurate representation of the result. Therefore, the RDF-star representation corresponding to the JSON in Listing 9 includes this information, as depicted in Listing 10.

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.[15] 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 11, 12 and 13 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 (which 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 that generates the RDF-star graph in Listing 14 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.

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

Table 1

Results of different reification approaches for the use cases with Morph-KGC^{star}. Materialization time in seconds, number of generated triples and characteristics of the mapping documents (number of triples maps and predicate-object maps (POM)) for the SemMedDB and SoMEF use cases with different reification alternatives

	SemMedDB				SoMEF			
	Mapping		Generation Time (s)	Number of Output Triples	Mapping		Generation Time (s)	Number of Output Triples
	TriplesMap	POM			TriplesMap	POM		
RML-star	10	10	2175.44	36,322,828	78	122	1084.73	15,102
Singleton Property	10	15	1937.86	86,983,087	78	158	1124.95	16,015
Std. Reification	9	20	2996.04	127,697,142	39	199	1201.36	21,268

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. As mapping partitioning [4] has not yet been extended for RML-star, we obtained the generation times without this optimization to fairly compare the different reification approaches. 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 prediction (i.e., entities and predications). All mappings used for the validation are openly available [9]. However, the data in the latter use case is licensed under the UMLS – Metathesaurus License Agreement,[16] which does not allow its distribution, but it may be accessed by obtaining an account with the UMLS license.[17]

Table 1 describes 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 requires 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 and singleton properties are the approaches that result in faster materialization, obtaining similar execution times for both datasets.

5.2.2. *Comparison with SPARQL-anything*

We also compare our proposed implementation of RML-star with SPARQL-Anything. To the best of our knowledge, SPARQL-Anything is the only SPARQL-based tool able 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 10 from the JSON file in Listing 9 using RML-star (Listing 15) and SPARQL-Anything (Listing 16).

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 [9]. 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 an SSD SAS Read-Intensive 12 GB/s.

SPARQL-Anything is not able to generate the RDF-star graph for SemMedDB, as it produces an out-of-memory error. SPARQL-Anything [6] has well-known scalability issues, e.g., taking a long time to produce results or hitting memory limits. On the contrary, Morph-KGC^{star} generates the 36M output triples in less than 1 hour.

[16] https://www.nlm.nih.gov/research/umls/knowledge_sources/metathesaurus/release/license_agreement.html

[17] An account with the UMLS license can be requested at https://www.nlm.nih.gov/databases/umls.html.

Table 2

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

	SemMedDB		SoMEF (separated 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}	2175.44	36,067,636	1084.73	15,102	13.86	14,821
SPARQL-Anything	Out of memory	Out of memory	630.18	15,155	Timeout	Timeout

Regarding SoMEF, we consider the case of a separate file for each GitHub repository (separated files) and the case of a single file with all the repositories (aggregated file), consisting on a JSON array of 237 objects.[18] SPARQL-Anything results in faster execution times for separated files, but Morph-KGC^{star} outperforms it for the aggregated file. Morph-KGC^{star} processes the aggregated file in a few seconds, while SPARQL-Anything is not able to generate the output after 48 hours. These results show that SPARQL-Anything performs better for small input data sources, while Morph-KGC^{star} can scale to large volumes of data that SPARQL-Anything is not able to process. We also noticed that, when empty string values appear in the input data, SPARQL-Anything generates triples with empty string literals. This is different from the behaviour of Morph-KGC^{star}, which does not generate triples in these cases. As a result, the knowledge graph generated by SPARQL-Anything contains a higher number of triples than the one generated by Morph-KGC^{star}. While this causes an inconsistency between the number of generated triples, deciding whether to generate terms for empty values is an open issue and, currently, different implementations handle them following ad-hoc approaches.

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 [44], developed community-driven tools for qualifying statements[19] (i.e., adding qualifiers to annotate a triple). Another approach is RDF-star [19], which has been gaining popularity and adoption by the community (e.g., it has been implemented by GraphDB, Apache Jena, Stardog, etc.) as a means of representing reified triples.

Mapping languages establish 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 [5,43,45]. The W3C's R2RML [13] focuses on transformations from relational databases to RDF. Extensions of this language were developed to overcome its limitations and broaden its capabilities [43]. Among these languages, we highlight RML [28], 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 [43]. For instance, SPARQL-Generate [32] and SPARQL-Anything [6] extend the query language SPARQL [40], whereas ShExML extends the constraints language ShEx [39].

So far, two declarative mapping languages have been proposed to generate RDF-star graphs from heterogeneous data sources based on R2RML. RML-star [14] extends RML for which this paper contributes Morph-KGC^{star} as an implementation. The other is R2RML-star [42], an extension over R2RML, for which an algorithm to translate SPARQL-star into SQL queries is provided. Unfortunately, the implementation of R2RML-star is not publicly available, and, at the time of writing, the permanent URL for the R2RML-star's ontology[20] 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 of R2RML-star is not openly available, its

[18] It must be noted that the number of triples is different with respect to separated files because duplicated triples generated from different repositories are removed.

[19] https://www.wikidata.org/wiki/Help:QuickStatements

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

comparison with the rest of the languages and associated tools is based on its description [42]. 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 support joins and recursion. The R2RML-star extension enables recursion, but joins can only be performed with R2RML views [3]. This occurs because the ranges of `star:subject` and `star:object` are `rr:ObjectMap`[21] 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 triples need to be generated, an additional `rml: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 [46] for virtual RDF-star graphs [42], 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 graphs with our engine is faster than standard reification, and similar to singleton properties. Finally, we compare our approach against 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. Given the increasing adoption of RDF-star by the Semantic Web community (e.g., graph stores, libraries or the RDF-star Working Group) and the lack of tools to generate RDF-star, 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 introduced in RDF 1.2 [20].

Our future work includes implementing the remaining modules of the new RML specification, and providing support to user-friendly mappings (YARRRML [25,27]). We also plan to improve the performance of RDF-star materialization, e.g., by extending mapping partitioning [4] to RML-star.

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

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. The work also received partial financial support from the Galician Ministry of Education, University and Professional Training, and the European Regional Development Fund (ERDF/FEDER program) through grants ED431C2018/29 and ED431G2019/04. Daniel Garijo is supported by the Madrid Government (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, in the context of the VPRICIT, and through the call Research Grants for Young Investigators from Universidad Politécnica de Madrid. Anastasia Dimou and David Chaves-Fraga are also supported by Flanders Make.

Appendix A. RML-star and SPARQL-Anything mappings

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

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

:descriptionTM a rml:AssertedTriplesMap ;
  rml:logicalSource :ls ;
    rml:subjectMap :soft ;
    rml:predicateObjectMap [
        rml:predicate sd:description ;
        rml:objectMap [ rml:reference "description.excerpt" ] ] .

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

Listing 15. RML-star mapping to create the RDF-star graph in Listing 10 from the JSON file in Listing 9

```
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 16. SPARQL-Anything snippet to create the RDF-star graph in Listing 10 from the JSON file in Listing 9

References

[1] Apache Software Foundation, Apache Jena, 2023, https://jena.apache.org.

[2] J. Arenas-Guerrero, morph-kgc/morph-kgc, Zenodo 2023. doi:10.5281/zenodo.5543552.

[3] J. Arenas-Guerrero, A. Alobaid, M. Navas-Loro, M.S. Pérez and O. Corcho, Boosting knowledge graph generation from tabular data with RML views, in: *Proceedings of the 20th Extended Semantic Web Conference*, Springer Nature, Switzerland, 2023, pp. 484–501. ISBN 978-3-031-33455-9. doi:10.1007/978-3-031-33455-9_29.

[4] 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* **15**(1) (2024), 1–20. doi:10.3233/SW-223135.

[5] 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, Proceedings of the 2nd International Workshop on Knowledge Graph Construction, Vol. 2873, CEUR Workshop Proceedings, 2021. ISSN 1613-0073, http://ceur-ws.org/Vol-2873/paper11.pdf.

[6] 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, *ACM Transactions on Internet Technology* **23**(1) (2023). doi:10.1145/3555312.

[7] M. Barker, N.P. Chue Hong, D.S. Katz, A.-L. Lamprecht, C. Martinez-Ortiz, F. Psomopoulos, J. Harrow, L.J. Castro, M. Gruenpeter, P.A. Martinez and T. Honeyman, Introducing the FAIR Principles for research software, *Scientific Data* **9**(1) (2022). doi:10.1038/s41597-022-01710-x.

[8] M. Bayer, 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.

[9] D. Chaves, A. Iglesias and D. Garijo, Zenodo, 2022, oeg-upm/rdf-star-generation: v1.0. doi:10.5281/zenodo.6919707.

[10] D. Chaves, A. Iglesias, D. Garijo and J.A. Guerrero, Zenodo, 2022, kg-construct/rml-star-test-cases: v1.1. doi:10.5281/zenodo.6518802.

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

[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. doi:10.3233/SSW210035.

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

[14] 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, International Semantic Web Conference, P&D, Vol. 2980, CEUR Workshop Proceedings, 2021. ISSN 1613-0073, http://ceur-ws.org/Vol-2980/paper374.pdf.

[15] 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:10.1016/j.websem.2009.07.004.

[16] 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. doi:10.1007/978-3-031-06981-9_20.

[17] 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:10.7717/peerj-cs.318.

[18] 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, Zenodo, 2022. doi:10.5281/zenodo.6845245.

[19] O. Hartig, Foundations of RDF* and SPARQL* (An Alternative Approach to Statement-Level Metadata in RDF), Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Vol. 1912, CEUR Workshop Proceedings, 2017. ISSN 1613-0073, http://ceur-ws.org/Vol-1912/paper12.pdf.

[20] O. Hartig, P.-A. Champin and G. Kellogg, RDF 1.2 Concepts and Abstract Syntax, W3C Working Draft, World Wide Web Consortium, 2023, https://www.w3.org/TR/2023/WD-rdf12-concepts-20230615/.

[21] 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.

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

[23] D. Hernández, A. Hogan and M. Krötzsch, Reifying RDF: What works well with Wikidata? Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems, Vol. 1457, CEUR Workshop Proceedings, 2015, pp. 32–47, ISSN 1613-0073, http://ceur-ws.org/Vol-1457/SSWS2015_paper3.pdf.

[24] 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 International Publishing, 2019, pp. 162–173. ISBN 978-3-030-21395-4. doi:10.1007/978-3-030-21395-4_12.

[25] 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.

[26] A. Iglesias-Molina, J. Arenas-Guerrero, T. Delva, A. Dimou and D. Chaves-Fraga, RML-star, W3C Draft Community Group Report, 2023, https://w3id.org/rml/star/spec.

[27] A. Iglesias-Molina, D. Chaves-Fraga, I. Dasoulas and A. Dimou, Human-friendly RDF graph construction: Which one do you chose? in: *Proceedings of the 23rd International Conference on Web Engineering*, Springer Nature, Switzerland, 2023, pp. 262–277. ISBN 978-3-031-34444-2. doi:10.1007/978-3-031-34444-2_19.

[28] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga and A. Dimou, The RML ontology: A community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF, in: *Proceedings of the 22nd International Semantic Web Conference*, Springer Nature, Switzerland, 2023, pp. 152–175. ISBN 978-3-031-47243-5. doi:10.1007/978-3-031-47243-5_9.

[29] A. Kelley and D. Garijo, A Framework for Creating Knowledge Graphs of Scientific Software Metadata, *Quantitative Science Studies* (2021), 1–37. doi: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] O. Lassila and R.R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, World Wide Web Consortium, 1999, https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[32] 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.

[33] 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.

[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] 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.

[36] 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.

[37] 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.

[38] T. Pellissier Tanon, Oxigraph, Zenodo, 2023. doi:10.5281/zenodo.7408022.

[39] 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.

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

[41] M. Rodríguez-Muro and M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, *Journal of Web Semantics* **33** (2015), 141–169. doi:10.1016/j.websem.2015.03.001.

[42] L. Sundqvist, Extending VKG Systems with RDF-Star Support, 2022, https://ontop-vkg.org/publications/2022-sundqvist-rdf-star-ontop-msc-thesis.pdf.

[43] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester and A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, *Journal of Web Semantics* **75** (2023), 100753. doi:10.1016/j.websem.2022.100753.

[44] D. Vrandečić and M. Krötzsch, Wikidata: A free collaborative knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85. doi:10.1145/2629489.

[45] 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.

[46] 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*, Springer International Publishing, 2020, pp. 259–277. ISBN 978-3-030-62466-8. doi:10.1007/978-3-030-62466-8_17.