Knowledge Graphs: Semantics, Machine Learning, and Languages
M. Acosta et al. (Eds.)

doi:10.3233/SSW230008

Evaluating Reification with Multi-Valued Properties in a Knowledge Graph of Licensed Educational Resources

Manoé KIEFFER [a,1], Ginwa FAKIH [a] Patricia SERRANO ALVARADO [a]
[a] *LS2N, UMR 6004, Nantes Université, 44300 Nantes, France*

Abstract. This paper presents the construction of a Knowledge Graph (KG) of Educational Resources (ER), where RDF reification is essential. The ERs are described based on the subjects they cover considering their relevance. RDF reification is used to incorporate this subject's relevance. Multiple reification models with distinct syntax and performance implications for storage and query processing exist. This study aims to experimentally compare four statement-based reification models with four triplestores to determine the most pertinent choice for our KG. We built four versions of the KG. Each version has a distinct reification model, namely standard reification, singleton properties, named graphs, and RDF-star, which were obtained using RML mappings. Each of the four triplestores (Virtuoso, Jena, Oxigraph, and GraphDB) was setup four times (except for Virtuoso, which does not support RDF-star), and seven different SPARQL queries were experimentally evaluated. This study shows that standard reification and named graphs lead to good performance. It also shows that, in the particular context of the used KG, Virtuoso outperforms Jena, GraphDB, and Oxigraph in most queries. The recent specification of RDF-star and SPARQL-star sheds light on statement-level annotations. The empirical study reported in this paper contributes to the efforts towards the efficient usage of RDF reification. In addition, this paper shares the pipeline of the KG construction using standard semantic web technologies.

Keywords. Knowledge graph, RDF reification, multi-valued properties, query evaluation, educational resources.

1. Introduction

When teachers want to create a new course, they typically do a keyword search for (open) Educational Resources (ER) on the web to reuse and integrate into their course. While there are numerous valuable and relevant resources available (such as slides, videos, figures, text, code, etc.), many remain undiscovered because they are not well connected. Moreover, using these resources can present legal challenges if their licenses are not compatible with the course's license. These legal issues can create barriers for both the teacher and the institution hosting the course. Ideally, the process of analysing available resources to match a course plan and verifying licenses should not be time-consuming. In our project, the goal is to design a solution that can identify a minimal, relevant set of educational resources with licenses that can protect such set of resources, whether

[1]Corresponding Author: Manoé Kieffer, e-mail: Manoe.Kieffer@univ-nantes.fr.

or not the licenses are open. Our aim is to help teachers create content reusing relevant resources and without having to focus on licensing aspects.

ERs can be described by their title, authors, language, license, etc., as well as the subjects they cover. ERs' subjects can be numerous but not equally relevant for the ER. Some subjects are the main focus, while others are only mentioned briefly. Therefore, the relevance of each subject should be identified, and their relationship with each ER should be weighed accordingly. The best way to make ERs findable and reusable is to use the principles of the Linked Data. Semantic web technologies will allow a detailed description and interconnection of ERs/The recent specification of RDF-star and SPARQL-star sheds light on statement-level annotations. One of the first public work-drafts of RDF 1.2, introduces quoted triples as another kind of RDF term which can be used as the subject or object of another triple[2]. In our particular use case, statement-level reification will allow annotating with scores the relation of ERs and the subjects they treat. As the number of subjects can be important, this reified relation is a multi-valued property. Thus, efficiently dealing with multi-valued properties is important as well.

Multiple reification models with distinct syntax and performance implications for storage and query processing exist. The main objective of this work is to experimentally compare four statement-based reification models on four triplestores to determine the most pertinent choice for our KG.

The contributions of this paper are twofold: (i) a methodology to build four versions of a knowledge graph of ERs using statement-level reification, namely standard reification, singleton property, named graphs and RDF-star, and (ii) an empirical evaluation of four triplestores (Virtuoso, Jena, GraphDB, and Oxigraph) with a set of seven SPARQL query templates grounded with up to six different instances (26 instantiated queries).

The rest of this paper is organised as follows. Section 2 explains the methodology we follow to build the KG of ERs. Section 3 explains the pipeline used. Section 4 evaluates experimentally the reification models. Section 5 describes the related works. Finally, Section 6 outlines our future work and concludes.

2. Knowledge graph description

Our project, aims to empower teachers to facilitate the creation of licensable ERs based on existing ones. The resources in our KG comprise unstructured ERs (documents, videos, and audio files, etc.), which are semantically annotated with DBpedia resources. By means of a wikification process, relevant DBpedia concepts related to ERs are used to provide a comprehensive description of each resource. This section introduces the used ontology (Section 2.1), an explanation of the wikification process (Section 2.2), and statistics on our KG (Section 2.3).

2.1. Used ontology

Figure 1 depicts the used ontology. Consistent with the IEEE LOM standard (Learning Object Metadata)[3], we define ERs as LOM Learning Objects. The LOM standard suggests a range of properties to describe learning objects, using common vocabularies such as Dublin Core and FOAF (dct:title, dct:creator, dct:language, dct:licence, dct:format, foaf:name, etc.).

[2]https://w3c.github.io/rdf-concepts/spec/#section-triples
[3]http://data.opendiscoveryspace.eu/ODS_LOM2LD/ODS_SecondDraft.html

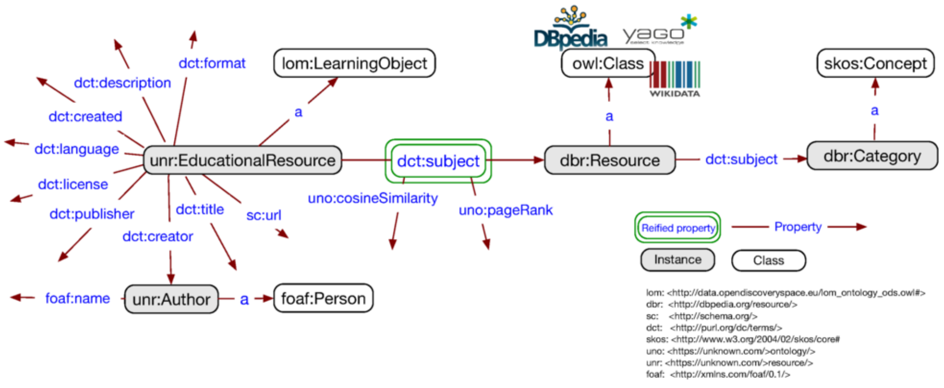


Figure 1. KG ontology.

The particularity of our ontology lies in the extension of the LOM description to consider the subjects treated in the learning objects with relevance scores. To do this, we use RDF statement-level reification. Reification allows making statements about statements in a generic manner. For us, it will allow to state that an ER treats a particular subject (in our case a DBpedia resource) *to some extent*. Concretely, it will allow to annotate the property dct:subject in a fact (unr:EducationalResource, dct:subject, dbr:Resource) with relevance scores. These scores are determined with a wikification process, which identifies pageRank and cosine similarity values in the range $[0..1]$. More information on this process is provided in the next section.

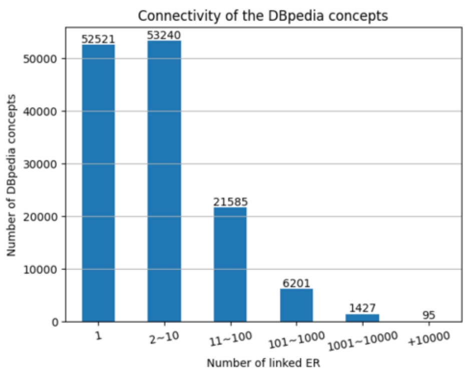
Besides being reified, dct:subject is a multi-valued property, i.e., a subject-predicate pair having several objects. These objects are DBpedia resources which are instances of classes in DBpedia, Wikidata, and Yago ontologies. We consider also DBpedia categories, which are used in Wikipedia to organize articles and pages by subject matter. Since the goal of our KG is to identify ERs based on their subjects, DBpedia categories are essential. DBpedia resources are associated with their categories through the dct:subject property (dbr:Resource, dct:subject, dbr:Category).

2.2. *Wikification of educational resources*

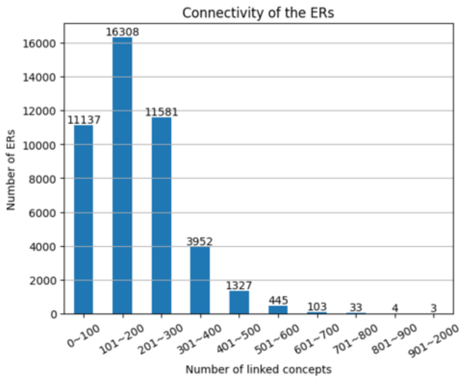
Entity linking techniques that map named entities to Wikipedia entities are called wikification. Text wikification is the task of automatically extracting the most important words and phrases in a document, and identifying for each such keyword the appropriate link to a Wikipedia article [1]. The wikification process generally involves two phases: term extraction and link disambiguation.

There are various approaches to wikification that differ in the techniques used for extracting phrases and linking them to external resources. The wikification tool called *Wikifier* has shown a good performance compared to some state-of-the-art approaches [2][4]. This tool identifies *mentions* - phrases extracted from the input document - and uses them as hyperlinks between Wikipedia pages. The Wikipedia pages linked by a mention are considered as candidate concepts for that mention. Wikifier constructs a bipartite graph consisting of the mentions and their corresponding candidate concepts. The internal structure of hyperlinks between Wikipedia pages is then leveraged to weigh the edges

[4]https://wikifier.org



(a) Connectivity of the DBpedia concepts.



(b) Connectivity of ERs.

Figure 2. Statistics of the connectivity in the graph.

(mentions) of the bipartite graph. A mention may have several candidate concepts because the same text can lead to different Wikipedia pages. To disambiguate mentions, the pageRank algorithm is applied over the graph. The concept with the highest pageRank score is selected for each mention, resulting in a set of Wikipedia concepts representing the input document. A threshold is then applied to retrieve the top-ranking concepts. In addition, Wikifier calculates the cosine similarity between the input document and the Wikipedia pages of the top-ranking concepts.

Currently, our project has collected a set of open educational resources that were wikified in the X5GON project[5].

2.3. Statistics and dataset content

There exist several vocabularies to describe RDF datasets. The VoID vocabulary [3] is the most well-established vocabulary.Additionally, the DCAT vocabulary [4] is a W3C recommendation to describe datasets, data services and data catalogs.We provide a description of our KG using VoID and DCAT in a VoID file[6]. The metadata for the dataset includes the label, license, SPARQL endpoint, provenance, prefixes used, and general statistics such as the number of triples, entities, subjects, objects, properties, etc. Our class partition currently consists of roughly 45K learning objects (lom:LearningObject), 13K authors (foaf:Person), twelve licenses (odrl:Policy), and 2,2M categories (skos:Concepts)[7] along with 135K DBpedia resources that serve as the reified concepts of ERs. It is important to note that reification is employed to annotate the dct:subject relation with two different annotations. The annotations are the pageRank score and cosine similarity score discussed in Section 2.2, those are numerical values between 0 and 1.

The distribution of the reified subjects (called concepts from now on) over ERs is far from being uniform. We consider the *connectivity of a concept* as the number of multi-valued properties (dct:subject) linking to it. Figure 2a shows that 100K over 135K

[5]https://www.x5gon.org

[6]https://gitlab.univ-nantes.fr/-/ide/project/clara/pipeline/tree/main/-/VoIDstatistics/clara-metadata.ttl/

[7]We obtained the entire hierarchy of DBpedia categories.

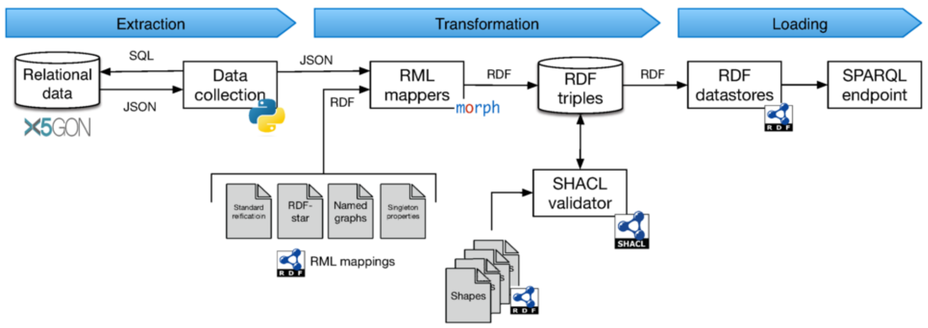


Figure 3. Pipeline of the ETL (Extract, Transform, Load) process.

concepts have poor connectivity. Roughly 52K concepts are associated with a single ER, while around 53K are connected to between 2 and 10 ERs, as shown in the first two columns. The third column indicates that around 21K concepts connect between 11 and 1000 ERs. Lastly, the last column shows that 95 concepts have very high connectivity, being used in more than 10K ERs.

The distribution of concepts by ER is shown in Figure 2b. This distribution corresponds to the multi-valued property dct:subject. The first three columns show that the majority of ERs are associated with less than 300 concepts (less than 300 values for the property dct:subject for the same ER). On the other hand, the last three columns show that only a small number of ERs (40 ERs) are linked to a high number of concepts.

The pageRank score of a concept is local to an ER, it depends on the number of concepts that the Wikifier associates with this ER. The sum of the pageRank values of all concepts linked to an ER is 1. Thus, the greater the number of concepts, the lower their pageRank score.

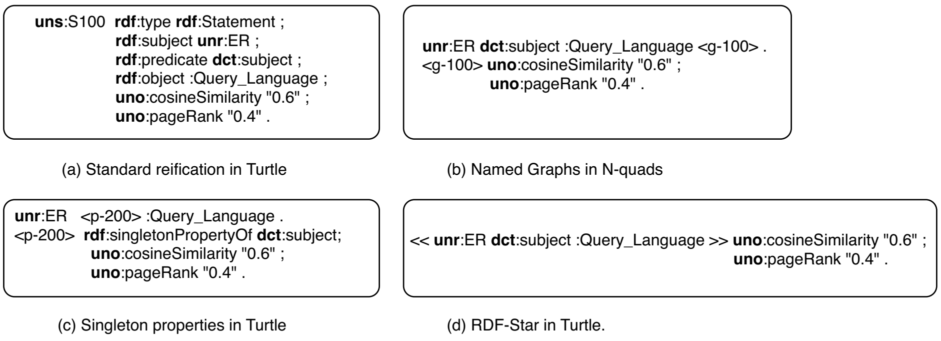
3. Data transformation pipeline

Figure 3 shows the pipeline for our ETL process. All related files can be found in the pipeline repository[8]. The extraction phase involves collecting data from a Postgres database. The data is extracted as JSON files because one of the attributes of the Postgres database contains JSON data, thus converting everything to JSON was deemed more efficient. In the transformation phase, the JSON files are converted into semantic RDF triples. To compare the different RDF reification models, four RML mappings were created in order to obtain standard reification, singleton properties, named graphs, and RDF-star. In particular, RML-star [5] is used to generate RDF-star data. SHACL is then used to validate our RDF graphs. In the loading phase, 15 different docker containers were loaded and setup using docker-compose. Jena, GraphDB, and Oxigraph, have one KG instance by reification model. Virtuoso has only 3 KG instances because it does not supports RDF-star. The rest of this section focuses on explaining the four different reification models, and then it explains the RDF-star mappings.

3.1. *Reification models*

Standard reification. The standard reification model was proposed within RDF primer standardised by W3C [6][9]. In this model, rdf:Statement is used to define the triple that

[8] https://gitlab.univ-nantes.fr/clara/pipeline
[9] https://www.w3.org/TR/rdf-primer/#reification



```
uns:S100  rdf:type rdf:Statement ;
          rdf:subject unr:ER ;
          rdf:predicate dct:subject ;
          rdf:object :Query_Language ;
          uno:cosineSimilarity "0.6" ;
          uno:pageRank "0.4" .
```

(a) Standard reification in Turtle

```
unr:ER dct:subject :Query_Language <g-100> .
<g-100> uno:cosineSimilarity "0.6" ;
          uno:pageRank "0.4" .
```

(b) Named Graphs in N-quads

```
unr:ER   <p-200> :Query_Language .
<p-200>  rdf:singletonPropertyOf dct:subject;
         uno:cosineSimilarity "0.6" ;
         uno:pageRank "0.4" .
```

(c) Singleton properties in Turtle

```
<< unr:ER dct:subject :Query_Language >> uno:cosineSimilarity "0.6" ;
                                          uno:pageRank "0.4" .
```

(d) RDF-Star in Turtle.

Figure 4. Syntax of the four reification models.

will be annotated (rdf:subject, rdf:predicate, and rdf:object). The defined statement can be identified by a blank node or a URI. Figure 4(a) gives the representation in RDF triples. It displays two score values annotated on a statement.

Named graphs. Carroll et al. [7] proposed an extension to the RDF data model that allows RDF graphs to be named by URIs, which are referred to as named graphs. In this approach, a named graph is represented as a pair (g, n), where g is an RDF graph and n is an IRI, a blank node, or a default graph. The statements to be annotated are defined in one RDF graph, while the annotations themselves are defined in another RDF graph. The annotations are directly linked to this graph. Figure 4(b) shows the syntax for this example.

Singleton properties. The singleton property model [8] proposes creating a unique property for every triple that has associated metadata. In this model, a new node is created to represent the new property, which is connected directly to the original annotation property using the proposed property *singletonPropertyOf*. The same property is used for all metadata associated with a statement. An example of RDF triples using this model is shown in Figure 4(c).

RDF-Star. [9][10] proposed RDF-star and SPARQL-star as extensions to RDF and SPARQL to enable graph nesting and simplify the representation of reified statements. RDF-star and SPARQL-star allow for the recursive nesting of graphs, eliminating the need for declaring edge identifiers that are linked with metadata. RDF-star enables the nesting of triples within other triples as subjects or objects by using double angle brackets $\ll \gg$. As a result, every reified statement can be interpreted as a single RDF triple. An example of RDF-star reification is shown in Figure 4(d).

3.2. Mapping JSON to RDF

The aggregate of the JSON files used as input to generate the four versions of our KG can be seen in Figure 5. Each ER is described by an id, a title, a description, etc. It can have several authors and can be associated to several concepts.

We define four mappings, one for each reification model. We use the RML language [10] to generate the RDF triples for standard reification, singleton property and named graph. To generate RDF-star we use the RML-star [5] language. Listing 1 shows an excerpt of the RML-star mapping used to transform the JSON data into RDF-star. This

[10]https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html

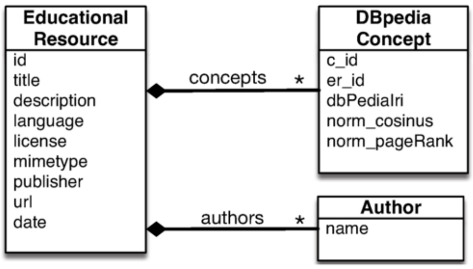


Figure 5. JSON aggregate of the source files for the RML mappings.

excerpt contains the RML-star rule that generates the triple with the reified property dct:subject (Lines 1 to 16) and the rule to generate one annotation (Lines 18 to 33). Lines 2 to 6 indicate an iteration over every ER in the JSON file. Lines 7 to 9 define the id of the ER as a subject for the triple. Lines 10 to 16 define the predicate for this subject (dct:subject) and multiple objects. These multiple objects are the set of concepts obtained from the JSON data. They are DBpedia resources treated as IRIs. Lines 19 to 23 again iterate over every ER in the JSON file. This time to generate the annotation. Lines 24 to 26 show that :ER_concept_link, that is the name of the first rule, is now the subject. Lines 27 to 33 define the annotation of the pageRank score, taken from the JSON under the attribute "norm_pageRank". In a similar way, the annotation for cosineSimilarity is generated.

We use Morph-KGC [11] as mapper to generate the RDF triples. Morph-KGC was able to generate the four versions of our KG through the four mappings including the RML-star mapping.

```
1   :ER_concept_link a rr:TriplesMap;
2       rml:logicalSource [
3           rml:source "json/ER/normal.json";
4           rml:referenceFormulation ql:JSONPath;
5           rml:iterator "$.resources[*]";
6       ];
7       rr:subjectMap [
8           rr:template "https://unknown.com/resource/{id}";
9       ];
10      rr:predicateObjectMap [
11          rr:predicate dct:subject;
12          rr:objectMap [
13              rml:reference "concepts.dbPediaIri";
14              rr:termType rr:IRI;
15          ]
16      ].
17
18  :ER_concept_context a rr:TriplesMap;
19      rml:logicalSource [
20          rml:source "json/ER/ER_0.json";
21          rml:referenceFormulation ql:JSONPath;
22          rml:iterator "$.resources[*]";
23      ];
24      rml:subjectMap [
25          rml:quotedTriplesMap :ER_concept_link;
26      ];
27      rr:predicateObjectMap [
            rr:predicate uno:pageRank;
            rr:objectMap [
                rml:reference "concepts.norm_pageRank";
                rr:datatype xsd:double;
            ]
        ].
```

Listing 1: Excerpt of the RML mapping for RDF-star reification.

4. Experimental evaluation of reification models

In our KG, the relation dct:subject is reified with two annotations (uno:pageRank and uno:cosineSimilarity). The number of values for this relation can be high (up to several thousands of concepts by ER, cf. Figure 2b) thus, the number of annotations by ER can be huge. The goal of this section is to compare four different reification models expressed over a multi-valued property. The triplestores we analyse are Virtuoso, Jena, GraphDB, and Oxigraph.

The rest of this section is organised as follows. First, Section 4.1 compares the four analysed reification models in terms of syntax and number of triples. Then, Section 4.2 describes the setup of our experiments. Section 4.3 shows the results of our experiments, comparing the size of the triplestores and the execution times of the different queries. Finally, Section 4.3 analyses the obtained results to extract our conclusion. All queries, the corresponding scripts, and the experiment results (raw execution times and plots) can be found in the queries_comparison repository[11].

4.1. Syntax comparison of analysed reification models

Described reification models differ in various criteria such as the total number of triples, flexibility, and syntax support.

Number of triples. Standard reification is the most costly approach since it needs five triples for each reified statement. Singleton properties needs three triples. Named graphs and RDF-star are the most compact models needing two and one triples respectively.

The second column of Table 1 shows the number of triples by reification model. In our KG, around 12M of triples are shared among all reification models. And more than 8M statements are reified. This amount of statements leads to the observed differences. As expected, RDF-star is the most compact model, followed by named graphs. The bulkiest model is standard reification followed by singleton properties.

Flexibility. All of these reification models are flexible when it comes to adding new annotations to an already reified statement. Adding new annotations only requires adding one additional triple for each approach. Additionally, all of these models cause no issues with multi-valued properties. Also, one advantage specific to named graphs is that reification can be defined also for a group of triples or even a dataset.

Syntax support. Standard reification and singleton properties conform to the core RDF model proposed in 2004. Named graphs represent an extension to the triple RDF model and is part of the standard RDF1.1, which was published in 2014. RDF-star proposes to extend the RDF specification further. Concerning the query language, all of these models are supported in the SPARQL standard, except for RDF-star which proposes SPARQL-star as a query language.

[11]https://gitlab.univ-nantes.fr/clara/queries_comparison

	Number of Statements	Virtuoso	Jena	GraphDB	Oxigraph
Standard reification	61,865,751	3.6 GB	52 GB	8.5 GB	6.5 GB
Singleton properties	45,335,637	3.6 GB	48 GB	257 GB	5.7 GB
Named graphs	37,071,104	3.2 GB	51 GB	6.4 GB	9 GB
RDF-Star	37,055,676	-	50 GB	6.6 GB	12 GB

Table 1. Generated DB size of different reification models.

The implementations of RDF-star[12] can follow three approaches: *PG (Property Graph), SA (Separate Assertion), or both*[13]. In the SA mode, quoted triples are not necessarily asserted in the graph. In the PG mode, any quoted triple is automatically asserted. Jena supports both modes, Oxigraph and GraphDB both support only the SA mode for RDF-Star. And Virtuoso does not support RDF-star at all.

4.2. Experimental setup

Experiments were run on a virtual machine with 128GB of RAM, 2GHz with 32 cores, on a Debian GNU/Linux 11 (Bullseye). All tests were run using docker images of the triplestores[14] [15] [16] [17]. All triplestores were parameterized with a query timeout of 30 minutes and given access to 16 GB of RAM. Only Oxigraph was not parameterized as we did not find the way to do it. GraphDB was also parameterized to use a context index when dealing with the named graphs version of the graph. All four triplestores were evaluated with the four reification models except for Virtuoso that does not support RDF-star, and GraphDB does not support singleton properties in an efficient way. This is because GraphDB makes the assumption that there will be only a small number of properties in the graph. This issue is described on the website of GraphDB[18]. Details on how exactly the experiments were run are given at the end of this section.

Query templates. Based on the series of queries A, B, and F used in [12], we define seven query templates that are presented as SPARQL-star queries for simplicity. The templates will be referred to as Q1 to Q7. All templates can be seen in Figure 6. Q1 and Q5 are grounded with instances of ERs while Q2, Q3, Q4, and Q6 are grounded with instances of concepts. Q7 is not grounded. In these queries, only the subjects are quoted. Q2, Q3, and Q4 are star-shaped queries.

- Q1 is a property path query. It returns the list of concepts and the associated hierarchy of categories (with skos:broader*) for a given ER.
- Q2 is a FILTER query that compares the annotations of a concept. Given a concept, it returns the associated ERs whose pageRank score is greater than the cosine similarity using the FILTER keyword.
- Q3 is a join query. It returns the set of ERs associated to three given concepts.
- Q4 is similar to Q3 but with a FILTER that specifies how the three pageRank scores must relate together.

[12]https://w3c.github.io/rdf-star/implementations.html
[13]https://w3c.github.io/rdf-star/cg-spec/editors_draft.html#sa-mode-and-pg-mode
[14]https://hub.docker.com/r/secoresearch/fuseki
[15]https://hub.docker.com/r/tenforce/virtuoso
[16]https://hub.docker.com/r/oxigraph/oxigraph
[17]https://hub.docker.com/r/khaller/graphdb-free
[18]https://graphdb.ontotext.com/documentation/10.0/devhub/rdf-sparql-star.html

Q1

```
SELECT ?concept ?categorie
WHERE {
  [ER] dct:subject ?concept .
  ?concept dct:subject/skos:broader* ?categorie .
}
```

Q2

```
SELECT ?er ?pr ?cosine
WHERE {
   << ?er dct:subject [concept] >> uno:pageRank ?pr ;
                                        uno:cosineSimilarity ?cosine .

   FILTER(?pr > ?cosine)
}
```

Q3

```
SELECT ?er ?pr1 ?pr2 ?pr3
WHERE {
   << ?er dct:subject [concept_1] >> uno:pageRank ?pr1 .
   << ?er dct:subject [concept_2] >> uno:pageRank ?pr2 .
   << ?er dct:subject [concept_3] >> uno:pageRank ?pr3 .
}
```

Q4

```
SELECT ?er ?pr1 ?pr2 ?pr3
WHERE {
   << ?er dct:subject [concept_1] >> uno:pageRank ?pr1 .
   << ?er dct:subject [concept_2] >> uno:pageRank ?pr2 .
   << ?er dct:subject [concept_3] >> uno:pageRank ?pr3 .

   FILTER( ?pr1 > ?pr2 && ?pr2 > ?pr3 )
}
```

Q5

```
SELECT ?title ?license ?format ?language ?publisher
?creator ?created ?description ?concept ?pr
WHERE {
  << [ER] dct:subject ?concept >> uno:pageRank ?pr .
  [ER]   dct:title ?title ;
       dct:license ?license ;
       dct:format ?format ;
       dct:language ?language ;
       dct:publisher ?publisher .

  OPTIONAL { [ER] dct:creator ?creator }
  OPTIONAL { [ER] dct:created ?created }
  OPTIONAL { [ER] dct:description ?description }
}
```

Q6

```
SELECT ?er ?pr
WHERE {
   { << ?er dct:subject [concept_1] >> uno:pageRank ?pr }
   UNION
   { << ?er dct:subject [concept_2] >> uno:pageRank ?pr }
   UNION
   { << ?er dct:subject [concept_3] >> uno:pageRank ?pr }
}
```

Q7

```
SELECT ?concept (COUNT(?er) as ?number_of_er)
WHERE {
   << ?er dct:subject ?concept >> uno:pageRank ?pr .

   FILTER(?pr > 0.01)
}
GROUP BY ?concept
```

Figure 6. Query templates

Q5 is join query that in addition uses the OPTIONAL clause. It returns all the information available for a given ER, the list of associated concepts, and the corresponding pageRank score.

Q6 is a UNION query that gets the ERs associated with one of 3 given concepts, using the UNION operator.

Q7 is a GROUP BY query that uses the COUNT operator. It returns the number of ERs by concept and it filters out the results lower than a pageRank threshold.

Groundings. Query templates are grounded with instances selected beforehand. We chose the groundings in order to evaluate the difference between multi-valued properties with few subjects or objects and multi-valued properties with a large number of subjects or objects. We use the multi-valued property dct:subject linking our ERs and their corresponding DBpedia concepts. This multi-valued property goes both ways, one subject to multiple objects, and one object from multiple subjects. We selected six ERs (the subject of the multi-valued property) and six concepts (the object of the multi-valued property). The first two ERs lead to a small number of multi-valued properties (3 and 7 objects), the next two have a medium number of multi-valued properties (108 and 270 objects), and the last two have a large number of multi-valued properties (1067 and 2053 objects). For DBpedia concepts, the first two concepts lead to a small number of multi-valued properties (3 and 13 subjects), the next two have a medium number of multi-valued properties (620 and 1123 subjects), and the last two have a large number of multi-valued properties (11486 and 21523 objects).

Methodology. Queries were instantiated with a number of groundings depending on the query as some queries need three different concepts (Q3, Q4, Q6). In that case only three grounding were given, each composed of three concepts. Instantiated queries were executed sequentially in increasing order of the size of the corresponding multi-valued property. Each query was executed 3 times sequentially. So, the capability of the triplestores to cache previous results had an important role. Queries were sent to the SPARQL endpoints with HTTP using a Python script.

4.3. *Experimental results*

This section first compares the size of the triplestores, then it explains the query execution times, and finishes with an analysis of obtained results.

Storage size

Columns 3 to 6 of Table 1 shows the size of the different triplestores with the different reification models. This table allows to see the differences in the storage size, but it also shows how efficiently the triplestores store each of the four reification models.

In general, Virtuoso uses the least amount of storage space and Jena uses the most. Storage costs do not change a lot across the reification models except for GraphDB that need 257 GB to store our KG with the singleton properties version. This observation was already put into light in [13]. The small difference of the storage volume by column (except for GraphDB using singleton properties) can be explained by the fact that an important part of the graph is common to all reification models.

Query execution results

A visualization of the execution times in seconds per query and triplestore is presented in Figures 7 and 8. Execution time is presented in a logarithmic scale. Each bar considers all corresponding groundings (except for Q1). The beginning of a bar indicates the fastest execution time and the top the largest. The black line is the average execution time and the yellow line is the median.

Figure 7(a) displays the results for Q1 which is a star property path query. Q1 is very challenging as it navigates multiple times (once for each concept linked to the grounded ER) through the dense hierarchy of DBpedia categories. Only the first four groundings were successfully executed. Queries with large groundings crashed. In general, Q1 has large execution times (sometimes even reaching 12 minutes for Oxigraph). We recall that GraphDB does not scale well using singleton properties so we were unable to experiment it with our KG. The overall observation is that Virtuoso outperforms for Q1 regardless of the reification model (with all averages being about 13 seconds), except for RDF-star, which is not supported.

Figure 7(b) displays the results for Q2 whose particularity is to compare two annotations of the same statements. The general observation is that Virtuoso outperforms the other triplestores except for named graphs where Oxigraph behaves very well. For standard reification, Virtuoso is the fastest (with an average of 0.16 seconds) followed by GraphDB (with an average of 0.38 seconds), then by Jena (with an average of 6.21

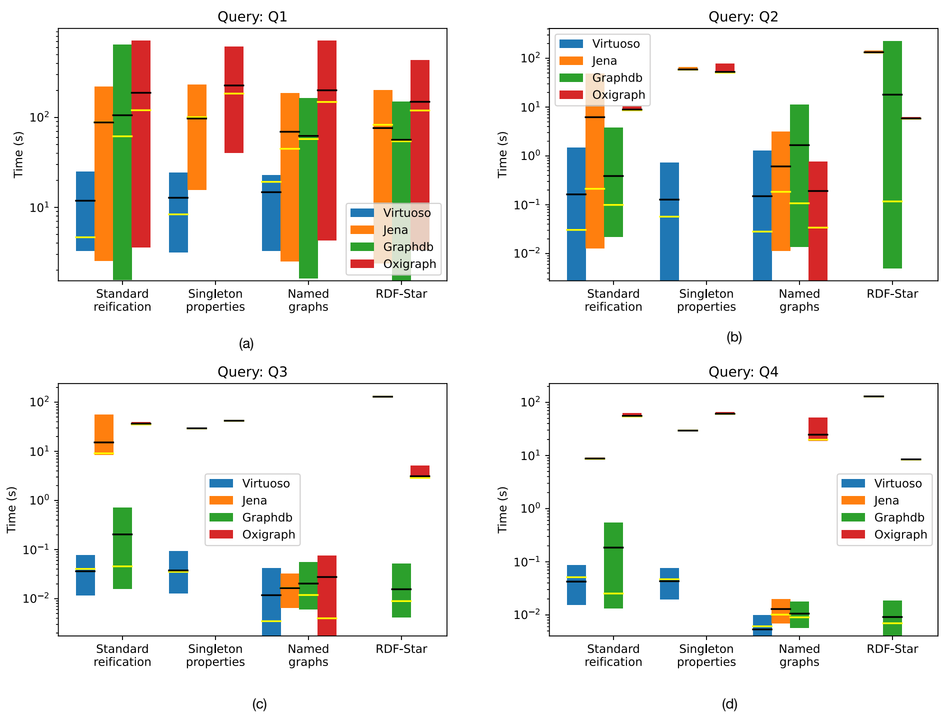


Figure 7. Comparison of Execution Time for Queries Q1-Q4 across Different Data Stores and Reification Approaches

seconds), and finally Oxigraph (with an average of 8.91 seconds). For singleton properties, like in Q1, only Virtuoso achieves good results. Jena and GraphDB have similar results (around 1 minute while Virtuoso achieves results on average less than 0.1 seconds). For named graphs, both Virtuoso and Oxigraph achieve similar results (around 0.1 second). They are followed by Jena, then by GraphDB. For RDF-star, once again GraphDB achieves good results followed by Oxigraph and Jena.

Figure 7(c) displays the results for Q3 that is a join query. Clearly, this figure shows that join queries are best executed over named graphs (all triplestores having an average of around 0.015 seconds). Concerning standard reification and singleton properties, Virtuoso outperforms the other three triplestores. For singleton properties again only Virtuoso achieves good results. For RDF-star, GraphDB achieves the best results (on average around 0.015 seconds). It is followed by Oxigraph, then far by Jena.

Figure 7(d) displays the results for Q4 that is a join query similar to Q3 but with a FILTER comparing the pageRank scores. Results are very similar to those of Q3. Execution times are higher than for Q3 but in general the cost of the FILTER is not very high, except for Oxigraph over named graphs. The average of Virtuoso, Jena, and GraphDB are around 0.01 seconds for named graphs, and Oxigraph's average is of 24.6 seconds. Again, GraphDB performs the best with RDF-star (in average 0.009 seconds).

Figure 8(a) displays the results for Q5. The singularity of this query is the use of the OPTIONAL clause. Again globally Virtuoso behaves the best (in average around 0.047

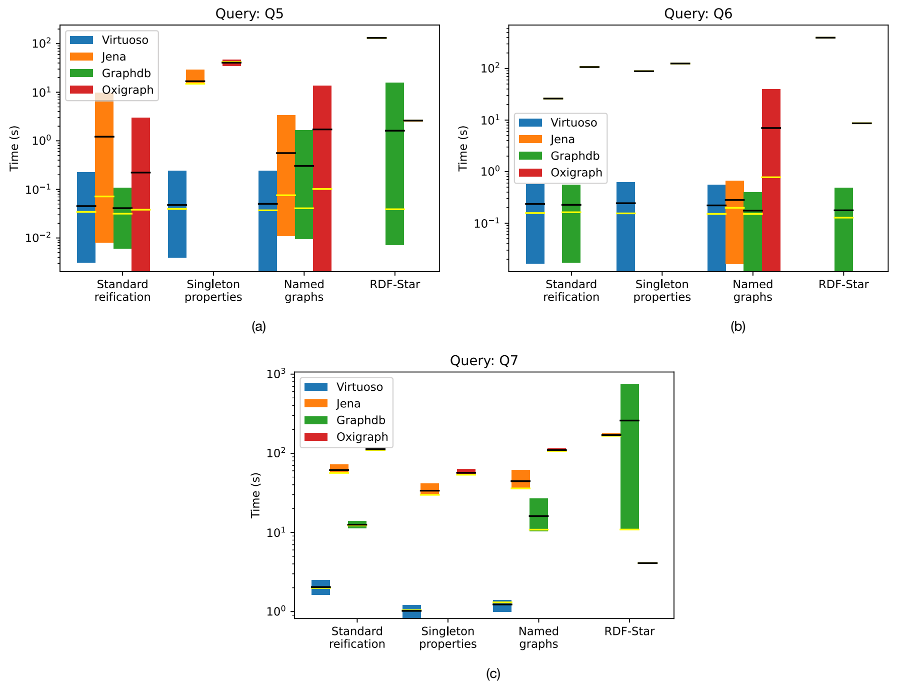


Figure 8. Comparison of Execution Time for Queries Q5-Q7 across Different Data Stores and Reification Approaches

seconds). For standard reification, GraphDB is as good as Virtuoso (with in average 0.041 seconds). GraphDB performs the best for RDF-star and Jena performs the worst.

Figure 8(b) displays the results of Q6. The particularity of this query is the use of the UNION clause. This query is globally well executed over named graphs (average of around 0.20 seconds) except for Oxigraph (in average of 6.99 seconds). In general GraphDB and Virtuoso behave the best. It is worth noting that again GraphDB executes well over RDF-star.

Figure 8(c) displays the results of Q7. The challenging aspect of this query is the GROUP BY operator. For that specific query, it is clear that Virtuoso has the best execution times regardless of the reification model (on average 1 or 2 seconds). The other triplestores are significantly slower (over 10 seconds), except Oxigraph with RDF-star (on average 4.09 seconds). Virtuoso is followed by GraphDB, then by Jena, and lastly by Oxigraph. It is worth noting that for Q7 Oxigraph behaves better than GraphDB on RDF-star.

Result analysis

The experimental evaluation done over our KG allows to draw the following conclusions. *Focus on reification models.* (a) Singleton properties is the least efficient reification model in our experiments. Only Virtuoso manages well this reification model. (b) Globally, standard reification and named graphs lead to good performances but named graphs is slightly better. (c) Even if in general RDF-star leads to bad execution times, frequently GraphDB obtains good results. (d) RDF-star is an elegant and compact model for statement-based annotations but triplestores should implement it more efficiently.

Focus on triplestores. (a) In general, Oxigraph is the least efficient triplestore in our experiments. However it is important to highlight that it performs better than Jena with RDF-star. (b) The execution times of Jena are consistently in second or third position but overall it outperforms Oxigraph. (c) In some experiments, GraphDB performs similarly to Virtuoso. (d) Virtuoso outperforms in most of our experiments but it should be noted that it does not support RDF-star.

The final conclusion for our KG is that the best choice would be Virtuoso with named graphs. Both standard reification and named graphs with Virtuoso exhibit similar performance. Named graph is slightly faster in some cases, in particular with join queries.

5. Related Works

Several works studied different reification methods and compared them according to several criteria. [13] focused on Wikidata and its representation in RDF using reification based on n-ary relations, standard reification, singleton properties and named graphs. Authors compared these models over five triplestores: 4store, Blazegraph, GraphDB, Jena, and Virtuoso. Their performance were measured based on 14 queries. Their results suggested that the singleton properties model was hardly supported but no other model was an outright winner. Concerning query performance, Virtuoso was the best followed by GraphDB and Blazegraph.

[14] realized an analysis of standard reification, named graphs, n-ary relations, singleton properties, companion properties (proposed in that paper) and RDF-star in its early stages. Experiments used Wikidata and DBpedia datasets on the triplestores Blazegraph, Stardog and Virtuoso. As DBpedia does not have singificant metadata, authors build a dataset with the Wikipedia revision history focusing on a company dataset. The experiments show that when the granularity of metadata is not by statement, companion properties and named graphs outperform. Concerning statement-level metadata, while standard reification results in the highest number of triples, it consumes the least storage in the database files and named graphs the most. This is because additional index structures for the graph identifiers are maintained. Concerning query performance, metadata characteristics have an impact on the reification models. Named graphs and RDF-star support queries against meta-metadata much better than the other models. In general, RDF-star can compete with named graphs if the metadata is on statement level. Moreover, both offer the best trade-off for mixed and data query workloads. In our experiments, queries do not contain data (i.e., triples) and we do not test querying meta-metadata.

[15] used three simple counting queries to analyse the internal representations of RDF-star in Stardog, Blazegraph and ExecuteSPARQLStar.[19] Experiments showed the divergence of the implementations of RDF-star when dealing with nested RDF-star statements. Blazegraph and ExecuteSPARQLStar behave similarly but Stardog was not able to deal correctly with nested RDF-star statements. That is because Stardog flattens the nested statements.

[16] proposed a data model called Labeled k-partite Graph (LKG) for storing and querying RDF triples with metadata. Authors compared experimentally LKG with Sin-

[19]https://github.com/RDFstar/RDFstarTools

gleton Property, RDF Reification, Named Graph and PaCE [17]. Used datasets were (with and without meta-knowledge): SPARQL Performance Benchmark (SP 2 Bench), the Biomedical Knowledge Repository (BKR), and the Gov-track. Results highlighed that LKG outperforms these methods by generating fewer statements, having a smaller graph size, avoiding resource redundancy, and achieving faster query response time.

[12] proposes a benchmark (dataset and set of queries) to analyse reification models. To illustrate the utility of the benchmark, authors analysed querying performance, storage efficiency and usability on the Stardog triplestore using three reification models: standard reification, singleton properties and RDF-star. Authors used the Biomedical Knowledge Repository (BKR) dataset[20] in order to make their results comparable with [8] that compares singleton property against standard reification. Twelve queries are used to evaluate performance. Five of these queries were proposed in this work to focus on SPARQL-star. Experimental results suggest that singleton property seems to have the worst performance. Probably because of the high number of unique properties and because indexes are usually not optimised with that in mind. Authors also observed that for simple queries, standard reification performs better than RDF-star. For complex queries, clearly, RDF-star outperforms standard reification.

[18] presents a novel approach for representing metadata, which outperforms existing reification models such as Singleton Property, Named Graph, PaCE, Companion Property, N-ary relations, RDF-star, and RDF-star [19]. The authors employed various datasets for their study, including a BKR dataset, a Gov-track dataset, a Synthetic dataset, and a dataset obtained from [20]. Through experiments, the proposed approach demonstrates advantages in handling multi-dimensional and nested metadata with reduced graph size and fewer generated statements.

6. Conclusion

This paper presented the pipeline for the generation of a knowledge graph (KG) of educational resources (ER) and the evaluation of several reification models with several triplestores. The objective was to identify the most suitable approach for this KG. To achieve this, we defined seven query templates instantiated in 26 grounded queries. Within the KG, reification was used in a multi-valued property to add two annotations whose range is between 0 and 1. Based on the insights derived from this experimental study, we were able to draw meaningful conclusions. Both, standard reification and named graphs with Virtuoso, exhibit similar performance. Named graphs show a slight advantage in some cases, in particular for join queries. RDF-star should be implemented more efficiently if quoted triples are included in RDF 1.2. Finally, for the KG presented in this paper, Virtuoso with named graphs, emerges as a good choice.

Acknowledgments

This work has received a French government support granted to the Labex Cominlabs excellence laboratory and managed by the National Research Agency in the “Investing for the Future” program under reference ANR-10-LABX-07-01. Authors thank Master students in Computer Science of Nantes University for her participation in some aspects of this work.

[20]https://zenodo.org/record/3894746#.ZAtFOS_pNpQ

References

[1] Mihalcea R, Csomai A. Wikify! Linking documents to encyclopedic knowledge. In: Conference on Information and Knowledge Management (CIKM); 2007. doi:10.1145/1321440.1321475.

[2] Brank J, Leban G, Grobelnik M. Semantic annotation of documents based on wikipedia concepts. Informatica. 2018;42(1):23-32.

[3] Alexander K, Cyganiak R, Hausenblas M, Zhao J. Describing Linked Datasets. In: Workshop on Linked Data on the Web (LDOW); 2009. .

[4] Albertoni R, Browning D, Cox S, Gonzalez-Beltran A, Perego A, Winstanley P, et al.. Data catalog vocabulary (DCAT)-version 2; 2020. Available from: https://www.w3.org/TR/vocab-dcat-2/.

[5] Delva T, Arenas-Guerrero J, Iglesias-Molina A, Corcho O, Chaves-Fraga D, Dimou A. RML-star: A declarative mapping language for RDF-star generation. In: International Semantic Web Conference (ISWC) Posters, Demos and Industry tracks; 2021. p. 5.

[6] Manola F, Miller E, McBride B, et al. RDF primer. W3C recommendation. 2004. Available from: https://www.w3.org/TR/rdf-primer/.

[7] Carroll JJ, Bizer C, Hayes P, Stickler P. Named graphs. Journal of Web Semantics. 2005;3(4):247-67. doi:10.1016/j.websem.2005.09.001.

[8] Nguyen V, Bodenreider O, Sheth A. Don't like RDF reification? Making statements about statements using singleton property. In: International World Wide Web Conference (WWW); 2014. p. 759-70. doi:10.1145/2566486.2567973.

[9] Hartig O. Foundations of RDF* and SPARQL* (An alternative approach to statement-level metadata in RDF). In: International Workshop on Foundations of Data Management and the Web (AMW); 2017. .

[10] Dimou A, Vander Sande M, Colpaert P, Verborgh R, Mannens E, Van de Walle R. RML: A generic language for integrated RDF mappings of heterogeneous data. Workshop on Linked Data on the Web (LDOW). 2014;1184.

[11] Arenas-Guerrero J, Iglesias-Molina A, Chaves-Fraga D, Garijo D, Corcho O, Dimou A. Morph-KGC star: Declarative generation of RDF-star graphs from heterogeneous data; 2022.

[12] Orlandi F, Graux D, O'Sullivan D. Benchmarking RDF metadata representations: Reification, singleton property and RDF. In: International Conference on Semantic Computing (ICSC); 2021. p. 233-40. doi:10.1109/ICSC50631.2021.00049.

[13] Hernández D, Hogan A, Krötzsch M. Reifying RDF: What works well with Wikidata? International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS). 2015;1457:32-47.

[14] Frey J, Müller K, Hellmann S, Rahm E, Vidal ME. Evaluation of metadata representations in RDF stores. Semantic Web Journal (SWJ). 2017;10:205—229. doi:10.3233/SW-180307.

[15] Orlandi F, Graux D, O'Sullivan D. How many stars do you see in this constellation? In: European Semantic Web Conference (ESWC), poster demo; 2020. p. 175-80. doi:10.1007/978-3-030-62327-2_30.

[16] Sen S, Malta MC, Katoriya D, Dutta B, Dutta A. Labeled k-partite Graph for Statement Annotation in the Web of Data. In: 2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). IEEE; 2020. p. 63-71. doi:10.1109/WIIAT50758.2020.00014.

[17] Sahoo SS, Bodenreider O, Hitzler P, Sheth A, Thirunarayan K. Provenance Context Entity (PaCE): Scalable provenance tracking for scientific RDF data. In: Scientific and Statistical Database Management: 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30–July 2, 2010. Proceedings 22. Springer; 2010. p. 461-70. doi:10.1007/978-3-642-13818-8_32.

[18] Sen S, Katoriya D, Dutta A, Dutta B. RDFM: An alternative approach for representing, storing, and maintaining meta-knowledge in web of data. Expert Systems with Applications. 2021;179:115043. doi:10.1016/j.eswa.2021.115043.

[19] Schueler B, Sizov S, Staab S, Tran DT. Querying for meta knowledge. In: Proceedings of the 17th international conference on World Wide Web; 2008. p. 625-34. doi:10.1145/1367497.1367582.

[20] Fu G, Bolton E, Rosinach NQ, Furlong LI, Nguyen V, Sheth A, et al. Exposing provenance metadata using different RDF models. arXiv preprint arXiv:150902822. 2015. doi:10.48550/arXiv.1509.02822.