RDF* and SPARQL*: An Alternative Approach to Annotate Statements in RDF

Olaf Hartig

Dept. of Computer and Information Science (IDA), Linköping University, Sweden
olaf.hartig@liu.se

1 Introduction

One of the major criticisms of RDF has been the lack of a convenient way to annotate data with metadata on a per-statement basis. Such annotations are a native feature in other contemporary graph data models (e.g., edge properties in the Property Graph model [7]) and there exist a number of popular use cases, including the annotation of statements with certainty scores, weights, temporal restrictions, and provenance information. To mitigate the inherent lack of a native support for such annotations in the purely triple-based data model of RDF, there exist several proposals to capture such annotations in the RDF context. However, these proposals have a number of shortcomings (cf. Section 2) and none of them has yet been adopted as a (de facto) standard.

We propose an alternative approach that is based on nesting of RDF triples and of query patterns. This approach allows for a more compact representation of data and queries, and it is backwards compatible with the existing approaches. In an ongoing research project we study the trade-offs of our proposal. With a poster in the conference we aim to introduce our proposal to the community and to present initial results of our research. In the remainder of this extended abstract we first discuss existing approaches to annotate statements in RDF; thereafter, we introduce our proposal, highlight the current status of our research, and provide an outlook on our future work.

2 Existing Approaches to Annotate Statements in RDF

As a running example, consider the two RDF triples in Figure 1a (represented in the standard Turtle syntax), which indicate the age of somebody named Bob. Assume we want to annotate the statement about the age—i.e., the second of these triples—with provenance metadata referring to the source and the creator of the statement.

A first approach, called *RDF reification*, is to apply the reification vocabulary as introduced in the RDF specification [4]. This approach requires us to include four additional triples to refer to the triple for which we want to provide metadata. The subject of these four additional triples has to be a new identifier (IRI or blank node) which, later on, may be used for providing the metadata. For instance, if we let a blank node labeled `_:s` be our new identifier, then Figure 1c lists the additional four triples required for our running example. Now, we may use the blank node to provide our example metadata as illustrated in Figure 1b. Note that for every triple for which we want to provide metadata we have to add the respective four reification triples to our dataset.

Then, to query so-represented metadata about statements using SPARQL, each query has to contain additional triple patterns to match the triples that establish the reification. For instance, assume we want to retrieve a list containing the name and the age of each person in our data and the respective sources of the statements about the persons' ages. To this end, we may use the SPARQL query in Figure 1d. Observe that the given query

```
:bob foaf:name "Bob";      _:s dct:creator <http://example.com/crawler1>;
     foaf:age 23.              dct:source <http://example.net/text.html>.
```

(a) (b)

```
_:s rdf:type rdf:Statement ;
    rdf:subject   :bob ;
    rdf:predicate foaf:age ;
    rdf:object    23 .
```

(c)

```
SELECT ?name ?age ?src WHERE {
   ?x foaf:name ?name ;
      foaf:age  ?age .
   ?r rdf:type rdf:Statement ;
      rdf:subject ?x ;
      rdf:predicate foaf:age ;
      rdf:object ?age ;
      dct:source ?src . }
```

(d)

```
SELECT ?name ?age ?src WHERE {
   ?x foaf:name ?name .
   ?x foaf:age  ?age .
   ?x    ?p     ?age .
   ?p rdf:singletonPropertyOf foaf:age .
   ?x dct:source ?src . }
```

(e)

```
SELECT ?name ?age ?src WHERE {
   GRAPH ?g1 {
      ?x foaf:name ?name .
   }
   GRAPH ?g2 {
      ?x foaf:age  ?age .
   }
   ?g2 dct:source ?src . }
```

(f)

Figure 1: Existing approaches to annotate statements in RDF: (a–c) data and (d) query for RDF reification, (e) query for singleton properties, and (f) query for named graphs.

contains four triple patterns to identify the triple(s) whose metadata we want to see. If we were also interested in potential metadata about the corresponding `foaf:name` triple, we would have to add another four reification-related triple patterns.

The example highlights two major shortcomings of RDF reification: First, adding four reification triples for every reified triple is inefficient for exchanging RDF data. Second, writing queries to access metadata about statements is cumbersome because any metadata-related (sub)expression in a query has to be accompanied by another subexpression to match the corresponding four reification triples.

To address these shortcomings other authors have proposed alternative approaches; the most notable of which include *singleton properties* [6], and an application of named graphs which, hereafter, we refer to as *single-triple named graphs* [5]. Figures 1e and 1f illustrate a corresponding version of the query in Figure 1d rewritten for singleton properties and for single-triple named graphs, respectively. As can be observed in these example queries, each of the two proposals still requires queries to contain verbose constructs whose only purpose is to match artifacts that the respective proposal introduces to establish the relationship between a triple and the metadata about it. An additional issue of the singleton properties proposal is that it introduces a large number of unique predicates, which is untypical for RDF data and, thus, disadvantageous for commonly-used SPARQL optimization techniques [5]. Another disadvantage of the proposal to use named graphs is that it inhibits an application of named graphs for other use cases.

In contrast to these approaches, our proposal in the next section allows for very concise queries; yet, it remains backwards compatible. That is, it can be implemented based on any system that has been designed and optimized for any of the other approaches. Additionally, our approach can also be mapped natively to a corresponding physical storage model as a foundation for novel implementations tailored to our proposal.

```
SELECT ?name ?age ?src WHERE {
 ?x foaf:name ?name .
 <<?x foaf:age ?age>> dct:source ?src.
}
```
(a)

```
SELECT ?name ?age ?src WHERE {
 ?x foaf:name ?name .
 BIND(<<?x foaf:age ?age>> AS ?t)
 ?t dct:source ?src . }
```
(b)

Figure 2: SPARQL⋆ query with (a) nested triple pattern and (b) new type of BIND clause.

3 A New Proposal: RDF⋆ and SPARQL⋆

The basis of our proposal is to extend RDF with a notion of nested triples. More precisely, this extension, which we call *RDF⋆*, allows for triples that represent metadata about another triple by directly using this other triple as their subject or their object. For instance, assume an extension of the Turtle syntax that implements the idea of nested triples by enclosing any embedded triple using the strings '<<' and '>>' (we call this extended syntax *Turtle⋆* and specify it in our technical report [3]). Then, all data and all metadata of our running example (i.e., Figures 1a–1c) may be represented as follows.

```
:bob foaf:name "Bob" .
<<:bob foaf:age 23>> dct:creator <http://example.com/crawlers#c1> ;
                     dct:source <http://example.net/listing.html> .
```

Given the outlined notion of RDF⋆ which supports (arbitrarily deep) nesting of triples, the crux of our proposal is to extend the SPARQL query language accordingly. That is, in the extended language, called *SPARQL⋆*, triple patterns may also be nested, which gives users a query syntax in which accessing specific metadata about a triple is just a matter of mentioning the triple in the subject (or object) position of a metadata-related triple pattern. For instance, by adopting the Turtle⋆ syntax as outlined above, we may represent the query in Figure 1d (as well as its rewritten versions in Figures 1e and 1f) in a more compact form as illustrated in Figure 2a. Moreover, an alternative, semantically equivalent form is to use an extended type of BIND clauses as demonstrated in Figure 2b. The latter example also highlights the fact that in SPARQL⋆, variables in query results may be bound not only to IRIs, literals, or blank nodes, but also to full RDF⋆ triples. For a detailed formalization of SPARQL⋆, including the complete extension of the full W3C specification of SPARQL, we refer to our technical report [3].

4 Discussion and Initial Results

We emphasize three orthogonal perspectives on our proposal: On one hand, RDF⋆ and SPARQL⋆ may be understood—and used—simply as syntactic sugar on top of RDF and SPARQL. In this sense, any RDF⋆-specific syntax such as Turtle⋆ may be parsed directly into plain RDF data that uses RDF reification or any of the other approaches to annotate statements in RDF. Likewise, SPARQL⋆ queries may be rewritten into ordinary SPARQL queries according to the selected statement-annotation approach. Then, based on such conversions of data and queries, our proposal may be supported easily by implementing a wrapper on top of any existing RDF triple store. An advantage of this implementation approach is not only the comparably small effort that it requires, but also the fact that such an implementation can readily benefit from possible optimizations that the triple store has for RDF reification (or any of the other related proposals).

As a formal foundation of such a wrapper-based implementation we have studied RDF⋆-to-RDF and SPARQL⋆-to-SPARQL mappings [2]. More specifically, we have defined a pair of such mappings that employs the RDF reification vocabulary, and we have shown formally that these mappings possess two desirable properties, namely the *information preservation* property and the *query result preservation* property [2].

On the other hand, our proposal may also be conceived of as a new abstract data model in its own right. Hence, this model presents a more feature-rich extension of the RDF data model and its query language SPARQL. As such, our proposal may be implemented natively by developing techniques to execute SPARQL* queries directly on a physical storage model that is designed to support RDF*. For instance, the idea of nested triples may be carried over to the physical level by means of a new storage model that embeds physical representations of triples into one another. Since such native implementations of our proposal may be tailored to leverage particular characteristics of RDF* and SPARQL*, they might be more efficient than wrapper-based ones.

Initial results towards such native implementations also comprise foundations only at this moment. That is, we have defined the RDF* data model and a formal semantics of SPARQL*, and we have shown properties related to redundancies in RDF* data [2].

Conceptually, our proposal extends RDF and SPARQL with a feature that is similar to the notion of edge properties in Property Graphs. Therefore, a third, more abstract perspective on our proposal is that it presents a step towards closing the gap between the RDF world and the world of graph databases. That is, our proposal may serve as the foundation of a conceptual mediator layer for integrating RDF data and Property Graphs. In fact, in addition to the aforementioned RDF*-to-RDF mappings [2], we also have preliminary results on reconciling RDF* and the Property Graphs model [1]. Hence, when combined, these works already provide a basis for integrating data across the different graph data models and for using SPARQL* as a common query language.

5 Outlook

So far we have focused on providing the formal foundations of our proposal. We consider establishing these foundations as the necessary preliminaries to systematically study the trade-offs of our proposal. Consequently, our future work is to conduct such a study, which will take multiple directions: First, we plan to investigate how appealing SPARQL* queries are to users in comparison to the corresponding SPARQL queries that would have to be written for other RDF-focused statement-annotation approaches (such as those in Section 2). Second, we aim to understand the *practical* consequences of executing SPARQL* queries based on a wrapper that employs our mappings. Third, and perhaps most interesting from a systems-research perspective, we want to investigate approaches to implement our proposal natively. A particularly interesting idea in this context is to carry over the notion of nested triples to the physical level.

References

1. O. Hartig. Reconciliation of RDF* and Property Graphs. *CoRR*, arXiv/1409.3288, 2014.
2. O. Hartig. Foundations of RDF* and SPARQL*. In *11th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW)*, 2017.
3. O. Hartig and B. Thompson. Foundations of an Alternative Approach to Reification in RDF. *CoRR*, abs/1406.3399, 2014.
4. P. J. Hayes and P. F. Patel-Schneider. RDF 1.1 Semantics. W3C Recommendation, Feb. 2014.
5. D. Hernández, A. Hogan, and M. Krötzsch. Reifying RDF: What Works Well With Wikidata? In *11th Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2015.
6. V. Nguyen, O. Bodenreider, and A. P. Sheth. Don't like RDF Reification? Making Statements about Statements Using Singleton Property. In *23rd Int. World Wide Web Conf. (WWW)*, 2014.
7. I. Robinson, J. Webber, and E. Eifrém. *Graph Databases*. O'Reilly Media, 2013.