An RDF Vocabulary for Meta-Programming with SPARQL Algebra

Miguel Ceriani[1,2] and Alejandro Vaisman[2]

[1] Sapienza, Università di Roma, Italy
ceriani@di.uniroma1.it
[2] Instituto Tecnológico de Buenos Aires, Argentina
mceriani@iitba.edu.ar, avaisman@itba.edu.ar

Abstract. SPARQL queries can be represented in RDF in order to work with them as first-class citizen in Semantic Web applications. The representation used so far, SPIN SPARQL, closely follows the abstract syntax tree, but fails to directly capture the semantics and is hard to work with for query rewriting. Here a novel vocabulary based on SPARQL Algebra is presented. The adequacy of the language is empirically shown through a few examples.

1 Introduction

SPARQL [2] is the standard query language for the Semantic Web and is defined on the RDF data model. It is used to access and modify remote data stores as well as local data, as part of the logic of a Semantic Web application. SPIN SPARQL syntax [1] has been proposed as an RDF representation of SPARQL. Such syntax allows to consistently store SPARQL queries together with the domain model. SPARQL expressions can thus be used together with RDF/OWL models to define complex constraints, rules, data visualizations, example queries, and so on. Furthermore, this triple-based representation opens the way to meta-programming techniques, in which this queries can be manipulated, rewritten, composed by the same means used for RDF.

The SPIN SPARQL syntax mimics quite closely the abstract syntax tree, providing an easy way to convert from and to the SPARQL textual syntax. Unfortunately the structural relationship between SPARQL syntax and semantics is not so immediate. As a fact, SPARQL semantics are defined in a two-step process. At first, a query is converted from its SPARQL syntax to another structure, composed with a set of operators that are called the SPARQL Algebra. The semantics are then operationally defined for the operators in the SPARQL Algebra. Our thesis is that the operations that have to do with query semantics would be simpler to express if the queries were represented by a vocabulary related to SPARQL Algebra instead of SPARQL syntax.

2 Related Work: SPIN SPARQL Syntax

The **SPIN SPARQL** vocabulary [3] is designed to represent a SPARQL query as a tree of RDF nodes. The top-level node of a query has type `sp:Query`, specifically is an

[3] The namespace is `http://spinrdf.org/sp#` and here is abbreviated with `sp:`

instance of one of its subclasses, such as `sp:Select` and `sp:Construct`. The elements in the `WHERE` clause (e.g., a `FILTER` clause or a triple pattern) are represented as instances of subclasses of `sp:Element` or `rdf:List` instances for grouping element lists. Listing 1 shows an example of a SPARQL query, while listing 2 shows its representation using SPIN SPARQL syntax.

Listing 1. Example of SPARQL Query

```
SELECT *
WHERE { OPTIONAL { ?p foaf:givenName ?name }.
        FILTER(?age > 18). ?p foaf:age ?age.
        OPTIONAL { ?p foaf:familyName ?surname }. }
```

Listing 2. Query of Listing 1 represented in SPIN SPARQL syntax (some details omitted)

```
[ a sp:Select;
  sp:where ( [ a sp:Optional;  sp:elements ( [ a sp:TriplePattern; ... ] ) ]
             [ a sp:Filter;    sp:expression [ ... ] ]
             [ a sp:TriplePattern; ... ]
             [ a sp:Optional;  sp:elements ( [ a sp:TriplePattern; ... ] ) ] ) ].
```

3 Proposed Syntax

We propose a new vocabulary to represent the SPARQL Algebra in RDF, available as RDF Schema at `http://meta-sparql.org/vocab/spa`. This URL is also the base URI for vocabulary elements, in this paper abbreviated as `spa:`. Each kind of operator in the SPARQL Algebra is represented by a corresponding class (e.g. `spa:BGP`, `spa:Filter`). The `spa:subOp` property connect an operator and its inputs (other operators). For the binary not commutative operators –`spa:LeftJoin` and `spa:Minus`, the property `spa:subOp` is specialized by the properties `spa:leftOp` and `spa:rightOp`. Classes and properties from SPIN SPARQL are used when applicable: to represent expressions, variables, and triple patterns/paths/templates.[4]

The proposed syntax has by definition the same expressive power of SPARQL Algebra, that in turn has the same expressive power of both SPARQL textual syntax and SPIN SPARQL. As SPARQL semantics are defined through SPARQL Algebra, this syntax is well suited when query semantics must be considered.

Listing 3. Query of Listing 1 represented in the new proposed syntax (some details omitted)

```
[ a spa:Filter;
  spa:filterExpression [ ... ];
  spa:subOp
    [ a spa:LeftJoin;
      spa:leftOp
        [ a spa:Join;
          spa:subOp [ a spa:BGP; spa:tuple [ a sp:TriplePattern; ... ] ],
                    [ a spa:BGP; spa:tuple [ a sp:TriplePattern; ... ] ] ];
      spa:rightOp [ a spa:BGP; spa:tuple [ a sp:TriplePattern; ... ] ]      ] ].
```

In Listing 2, the semantics of the query are not so evident. The elements (instances of `sp:Optional`, `sp:Filter` and so on) are apparently all on the same level and one

[4] Moreover, we require each variable to be identified by a unique RDF resource so that variable identity can be directly evaluated.

may be tempted to think that the order between them may be safely changed. Conversely, the opposite is true: the order of the sequence (represented as an `rdf:List`) is critical for the semantics of the query. That means that any task related to the behavior of the query needs to take in account the ordering of elements.

In Listing 3 the tree corresponds directly to the semantics of the query. There is no use of `rdf:List` or other generic sequences because the semantics are given by the structure of algebra operators. Meta-queries and transformations that deal with the meaning of queries can thus be more easily designed. In the next Section some empirical evidence of this advantage is presented.

4 Examples of Meta-SPARQL

We will show some examples in which our proposal is compared with SPIN SPARQL (the only other RDF representation of SPARQL to date). So far there is not much documented use of SPIN SPARQL for meta-programming, so the comparison is made through examples we designed for a couple of use cases. We will in all the cases imply that RDFS inferencing is in place to simplify the queries.

Studies have been carried on statistics to gather on queries in order to classify them somehow (usually to predict their performance when executed). The Linked SPARQL Queries Dataset (LSQ) [3] is specially noteworthy for the recollection of both query structure –represented through the SPIN SPARQL syntax– and statistics from logs of a variety of public SPARQL endpoints. The following queries (re)calculate one of the query statistics considered in LSQ, the number of Basic Graph Patterns (BGPs) in each query. Even if this one of the most relatively straightforward statistics, the complexity of using SPIN SPARQL syntax is apparent. For other statistics (e.g., variables that participate in a join) queries using the SPIN SPARQL syntax becomes more complex still, due to the number of different cases that must be considered.

Listing 4. Meta-query counting the BGPs in each query (SPIN SPARQL syntax)

```
SELECT ?query (COUNT(?tupleStartingBgp) AS ?numOfBgps)
WHERE { ?query ((sp:elements|sp:where)?/rdf:rest*/rdf:first)* ?tupleStartingBgp.
        ?tupleStartingBgp a sp:Tuple.
        FILTER NOT EXISTS {
          ?previousTuple ^rdf:first/rdf:rest/rdf:first ?tupleStartingBgp.
        } }
```

Listing 5. Meta-query counting the BGPs in each query (proposed syntax)

```
SELECT ?query (COUNT(?bgp) AS ?numOfBgps)
WHERE { ?query spa:subOps* ?bgp.
        ?bgp a spa:BGP.          }
```

As an example of query rewriting, we consider the use case of adding a filtering step to a query. From a representation of the original query, in both cases we need to create a new resource that refers to the original one and represents the added step.

Listing 6. Turtle code to add a filtering step to a query (SPIN SPARQL syntax)

```
ex:queryWithFilter a sp:Select;
                   sp:where ( [ a sp:SubQuery;
                                sp:query ex:query ]
                              [ a sp:Filter;
                                sp:expression ex:filter ] ).
```

Listing 7. Turtle code to add a filtering step to a query (proposed syntax)

```
ex:queryWithFilter a spa:Filter;
                   spa:subOp      ex:query;
                   spa:expression ex:filter.
```

The case using SPIN SPARQL adds quite a deal of representational complexity to the query due to the usage the subquery syntax. In the `spa` syntax only the related functional operator is added and the query may be –if possible– further simplified by additional UPDATE requests. As an example, the following UPDATE request may be used as a rule to join adjacent filters. In a similar way, rules to push filters down the tree when possible may be defined. Similar rules would be hard to write from SPIN SPARQL, unless preceded by a complex set of rules that modify the syntax tree to reflect the semantics (thus realizing in a way the conversion to SPARQL Algebra).

Listing 8. UPDATE request to join adjacent filters (proposed syntax)

```
DELETE { ?filter1 spa:expression ?expr1;
                  spa:subOp      filter2. }
INSERT { ?filter1 spa:expression [ a sp:and; sp:arg ?expr1, ?expr2 ];
                  spa:subOp      ?subop.                              }
WHERE  { ?filter1 a spa:Filter;
                  spa:expression ?expr1;
                  spa:subOp      ?filter2.
         ?filter2 a spa:Filter;
                  spa:expression ?expr2;
                  spa:subOp ?subop.      }
```

5 Conclusions and Future Work

This paper presented a vocabulary for representing SPARQL queries in RDF. Contrary to the existing SPIN SPARQL vocabulary that follows the SPARQL abstract syntax tree, this vocabulary –abbreviated as `spa`– follows the SPARQL Algebra, that is the one against which the SPARQL operational semantics are defined. The `spa` vocabulary may thus be used for all the use cases in which SPARQL queries have to be analyzed or transformed according to their semantics. Based on some common use cases, an empirical evalution is given.

This a first step in exploring RDF-based syntaxes for queries alternative to SPIN SPARQL. Having a syntax more amenable to meta-querying and meta-programming may encourage developers to experiment with these techniques, to the advantage of the whole Semantic Web programming community. Further evaluation may be carried on by more extensively identifying a set of use cases and formally evaluating the usage of this novel syntax.

References

1. Fürber, C., Hepp, M.: Using SPARQL and SPIN for data quality management on the semantic web. In: Business Information Systems. pp. 35–46. Springer (2010)
2. Harris, S., et al.: SPARQL 1.1 Query Language. W3C REC 21 March 2013
3. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.C.N.: LSQ: The Linked SPARQL Queries Dataset. In: Proc. of ISWC 2015. pp. 261–269. Springer (2015)