RDF Query Languages: A state-of-the-art

Greg Karvounarakis
Institute of Computer Science, FORTH,
Vassilika Vouton, P.O.Box 1385,
GR 711 10, Heraklion, Greece
and
Department of Computer Science,
University of Crete, GR 71409,
Heraklion, Greece

1. Introduction

Several approaches have been proposed, as far as querying RDF metadata is concerned, have been proposed. These approaches can be generally subdivided into two main categories:

- The SQL/XQL style approach, viewing RDF metadata as a relational or XML database
- Viewing the Web – described by RDF metadata – as a knowledge base and, thus, applying knowledge representation and reasoning techniques on RDF metadata.

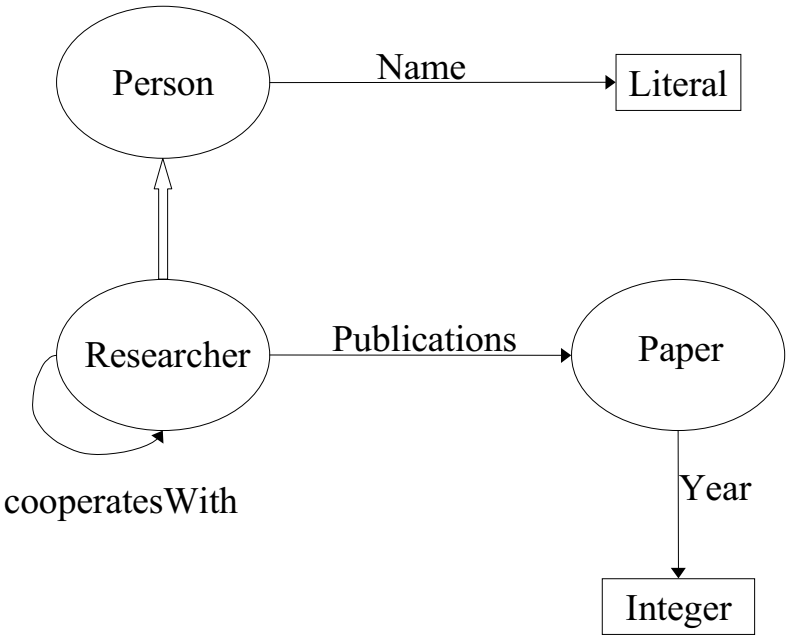

Figure 1: Simple RDF Schema graph

The simple RDF Schema shown in figure 1 will be used throughout this report, in order to clarify the application of the proposed languages to “real” RDF descriptions.

2. Database approach – RDF Query Specification (IBM)

This first approach, proposed by IBM, is based upon the RDF for XML Java package [1], created by the first. This tool allows one to create RDF objects, as defined in the context of this work, by reading XML encoded RDF metadata (using the syntax employed in the first phase of the creation of RDF). It then provides an API to perform several operations on objects of this class.

Based on this platform, they created a query language [2] in order to be able to query resources, described in these RDF instances. The main constructs of this language are:

- **Select, From**: as in common SQL for relational databases, these tags are used to define views of the result and declare the domain on which the query is performed. The “From” tag defines a container, as defined in the RDF Model, which consists of URIs to the metadata descriptions to be queried.

  Suppose that the URL `http://www/people#PersonsList` specifies a collection of Resources of Persons – as defined in the schema above. The simplest example could be the following:

```
<rdfquery>
      <rdfq:From eachResource=”http://www/people#PersonsList”/>
</rdfquery>
```

  This query returns all resources from the specified collection.

  We could also perform the following query (where the value of the “properties” attribute is a space-separated list of the names of the properties that should be selected in the projection):

```
<rdfquery>
      <rdfq:From eachResource=”http://www/people#PersonsList” >
            <rdfq:Select properties=”Name cooperatesWith”>
                  <rdfq:Property name="Publications”/>
            </rdfq:Select>
      </rdfq:From>
</rdfquery>
```

The result of this query is a list of resources that have a property called “Publications”, with the properties “Name” and “cooperatesWith” and their values.

- **Condition**: This tag is used to describe conditions on the values of properties (similar to the conditions in the “where” clause of common SQL.

Example:

```
<rdfquery>
    <From eachResource=”http://www/people#PersonsList” >
       <Select properties=”Name cooperatesWith”>
             <Condition>
                <equals>
                   <Property name="Name" />
                   <rdf:String>Neel Sundaresan</rdf:String>
                </equals>
             </Condition>
       </Select>
    </From>
 </rdfquery>
```

- **Path expressions**: This construct can be used in order to navigate across properties in an RDF graph.

  The following query can be used to get all resources, which have a property named “Publications”, the value of which has a property called “Year”, which satisfies the given condition.

```
<rdfquery>
   <From eachResource=”http://www/people#PersonsList” >
      <Select>
            <Condition>
                  <greaterThan>
                        <Property path=”Publications/Year”/>
                        <rdf:Integer>1995</rdf:Integer>
                  </greaterThan>
            </Condition>
      </Select>
   </From>
</rdfquery>
```

- **Union, Intersection and Difference**: These are used in order to perform the corresponding algebraic set operations on result sets.

  For example, one can issue the following query:

```
<rdfquery>
   <Union>
     <From eachResource="http://www/people/neel">
       <Select>
            <Property name="Publications"/>
       </Select>
      </From>
      <From eachResource="http://www/people/Ashok">
       <Select>
            <Property name="Publications"/>
```

```
        </Select>
      </From>
   </Union>
</rdfquery>
```

Union is used in order to join the results of the two queries in a set.

- **Group, Order**: These constructs allow as to group or sort results, according to the value of a property.

For example, we can group the resources returned by previous according to the year of publication:

```
<rdfquery>
   <From eachResource="http://www/people/neel">
      <Select>
         <Group>
            <Property path="Publications/Year"/>
         </Group>
      </Select>
   </From>
</rdfquery>
```

- **Quantifiers**: The quantifiers forAll and exists are defined in this language, in order to support queries based on the corresponding quantification operators.

If, for instance we want to get all those researchers that have at least one publication in 1998, we can issue the following query:

```
<rdfq:rdfquery>
   <rdfq:From eachResource="Almaden_Researchers">
      <rdfq:Select>
         <rdfq:Condition>
            <rdfq:Quantifier type="exists" var="x">
                  <rdfq:Property path="Publications"/>
                  <Condition>
                    <rdfq:equals>
                         <rdfq:Property var-ref="x"
                         name="Year"/>
                         <rdf:Integer>1998</rdf:Integer>
                      </rdfq:equals>
                  </rdfq:Condition>
            </rdfq:Quantifier>
         <rdfq:Condition>
      </rdfq:Select>
   </rdfq:From>
</rdfq:rdfquery>
```

The variable x takes all values from the collection (the resources which have a property named “Publications”) and then we use var-ref to reference the values of this variable.

Therefore, this QL provides all the constructs that are necessary in order to query specific documents that are described by specific schemata, that is we know in advance which properties each resource has, and what the relations between resources are. The problem in this approach is that it is not actually what one should expect from an RDF – or generally metadata – query language. It is actually a transformation of the XQL, in order to be able to query RDF descriptions. It is actually a language for querying data, as XQL instead of metadata. This should be considered normal, since the authors seem to view RDF just as an XML instance, not as a metadata model that can be encoded using XML syntax for interoperability purposes. This is obvious by the way they model RDF, according to the Document Object Model (DOM), which is used in order to model the structure of XML documents.

Moreover, this approach disregards RDF schema relationships, on which RDF instance documents are based, therefore losing a great deal of the semantics of the descriptions. For example, if there is a description of a “Person”, as defined in the schema above, and we know that a “Researcher” cooperatesWith them, we cannot infer that the “Person” is also a “Researcher”, as the range constraint in the schema implies. They only consider vocabulary specific inferencing as a future research direction and try to deal with primitive database types that could be supported in the future, which is more of an XML syntax matter than a metadata description problem.

3. Viewing the Web as a knowledge base (W3C related)

This second approach is mostly supported by the W3C RDF working group and other related researchers, as well as the founders of RDF itself. It seems that this is the approach of researchers coming from a different community (the knowledge representation community) but it could also be that this approach seems more likely to achieve the initial goals – for which RDF was initially proposed. Since the initial motivation that led to RDF was the need to represent *human-readable* but also *machine-understandable* semantics, it seems obvious that we should use this representation to do something clever with it. In this context, the following requirements for an RDF query language can be identified ([3],[4]):

- The underlying repository, in which the RDF descriptions are stored, should support the expressive power of the RDF model (assertions), as well as of the RDF Schemata (class and property hierarchies).
- The query language should abstract from the RDF syntax specifications, that is the XML encoding of RDF metadata.
- Such a query language should also provide several facilities – ranging from simple property-value queries, path traversal based on the RDF

graph model, to complex Datalog like queries. Several categories of deductions/inferences can be envisaged:

- Subsumption between classes and/or properties not explicitly stated in an RDF Schema using rdfs:subClassOf and rdfs:subPropertyOf.
- Classification of resources: if we know that the domain of a property is of a specific class, we could infer that a resource that appears in a metadata instance, having this property, is actually an instance of this class.
- Inverse references fulfilling: If, for example, we know that Researcher1 has Publication Paper1, then we could infer that Paper1 has author Researcher1.
- Automatic Query Expansion in order to explore generalization/ specialization (broader term, narrower term and synonym) relations between property values.

Based on these initial ideas and specifications, two query and inference languages for RDF have been proposed:

3.1. Query and Inferencing Service for RDF ([4], [5])

This approach is practically trying to achieve the aims outlined above, by mapping RDF metadata to Frame Logic. By loading the triples extracted from the RDF metadata into an F-logic knowledge base, according to some schema definitions extracted from the included RDF Schemata, they organize the RDF descriptions as a knowledge base. For example, our sample schema would be loaded in F-logic by the following F-logic statements (manually in the current implementation):

- Class definitions:

```
Person:: Object.
Researcher:: Person.
Paper:: Object.
```

- Attribute (property) definitions:

```
Person[Name=>> Literal].
Reseacher[Publications=>> Paper;
      CooperatesWith=>> Researcher].
Paper[Year=>> Integer].
```

For instance, in order to perform the query “give me all the resources that have a property ‘Publications’”, we should say in F-logic:

```
FORALL X, Y <- X[Publication->>Y].
```

Depending on schema definitions, one can perform at least basic inferencing on the RDF descriptions. For example, class hierarchies expressed in F-logic rules, allow inferencing based on sub-classing or refining queries by eliminating irrelevant information – according to the schema – that would otherwise match the query. Constraints in values

of properties can also be represented in this way, allowing such a type of validation and inferencing on incomplete information.

For example, according to the schema we know that:

```
Researcher:: Person[Publication=>>Paper].
```

Therefore, according to the semantics of F-logic, we can imply that all persons that have publications are instances of the class “Researcher”, even if this fact is not explicitly stated in the RDF descriptions.

Moreover, we can explicitly add inference rules. For example, we know that a Researcher only cooperatesWith another Researcher, so we can add the inference rule:

```
FORALL Person1, Person2
      Person1:Researcher[cooperatesWith->>Person2] <-
      Person2:Researcher[cooperatesWith->>Person1]
```

This means that if we know that Person2 is a Researcher and cooperatesWith Person1, then we can imply that Person1 is a Researcher and cooperatesWith Person2, even if this is not implicitly stated. Thus, if we then submit a query, requesting all instances of the class Researcher, according to this rule, we are also going to get all resources that a Researcher cooperatesWith.

Another type of inference proposed is automatic query expansion, as in the example of the Classification Scheme Mapping [5]. Moreover, F-logic syntax for the query language seems more user friendly, compared to the RDF encoding, while recursive queries can be performed using the same, F-logic style, query language on the resulting knowledge base. In a similar way, any deductive database (as is F-logic) can be used in the same way.

Finally, F-logic allows us to also perform some kind of queries on the schema. For example, one can request all Researchers that have a property with the value “Christophides”. In our case this would mean that either the name or the cooperatesWith property would have that value. This would be encoded in F-logic as:

```
FORALL Res, Prop, T <-
      Res:Researcher[Prop=>>T] AND
      Res[Prop->>”Christophides”]
```

The supposed advantage of SiLRI, the tool created by this working group at W3C, is that it provides a lightweight and portable implementation of the inference engine described above, including the functionality of F-logic. Therefore, it is probably suitable for mobile code (e.g. intelligent agent applications) implementations.

However, there are some problems with this approach. Firstly, not all RDF expressions can be directly expressed using F-logic. For example, no attempt has been done do describe RDF Containers, as bags etc, or reified statements can be represented, syntactically and semantically. Moreover, this approach is a common object-oriented one, not regarding properties as first-class objects, as the RDF model suggests. Consequently, no inference can be performed based on property hierarchies.

Furthemore, inference is based upon manually added inference rules; no attempt has been made to define the semantics of RDF itself. No generic inference facilities have been investigated, based on RDF/RDFS semantics, which would apply to any schema instance. What was actually the aim of this work was to be able to load RDF metadata annotations in Ontobroker [7, 8], which uses F-logic as its internal representation language, since the annotation language proposed by Ontobroker is not widely supported. Moreover, since Ontobroker presumes the existence of a predefined central ontology for each community, the need to be able to load individually created RDF Schema instances is less important to them; all information providers are likely to follow the organization of the ontology. Therefore, specific inference rules are part of the central ontology.

3.2. Metalog ([6])

This approach, also related to the W3C RDF working group, is similar at many points to the one described above. It attempts to allow inferencing to be performed on RDF metadata, using knowledge representation and reasoning techniques. However, this approach is closer to RDF model's *property-centric* approach, as properties in the RDF model are actually binary relations between resources.

The added value of this proposed system is that it abstracts even more the query language from the syntax, defining the query language specifications in a higher level. By identifying the correspondence between RDF statements (triples) and predicates in logic, one can view a query language in RDF at a logical layer, allowing any logical relationship to be expressed. What the Metalog approach suggests is an extended RDF Metalog schema, with the addition of logical connectors (and, or, not, implies) and variables to RDF Schema. This way, one can encode inference rules in RDF schema instances.

At this level, one can use the full expressive power of first-order predicate calculus in order to describe the schema of the query language. In order to create a computationally feasible implementation, one can then use a subset of the predicate calculus, as is logic programming. The implementation of Metalog suggests the use of Datalog. This approach has the obvious advantage of allowing the designers to directly address the matter that arises with the trade-off between expressive power and computational efficiency.

According to this approach, apart from the schema definition:

```
Publication(X,Y) => in(X, “Researcher”)
```

one can add rules in the schema as:

```
(Publication(X,Y) and Year(Y,Z)) => gotDegreeBefore(X,Z)
```

Then we are able to get all Researchers, who got their degree before a specified year:

```
GotDegreeBefore(X, 1990).
```

Another point that is highlighted in this approach is the need for a user-friendly interface for the query language, actually proposing the use of natural language to express queries. The language proposed maintains all the expressive power of the underlying Metalog schema, and also encapsulates inferencing facilities, together with the ability to express RDF descriptions of any kind.

However, no actual query language is proposed. What Metalog actually suggests is a way to embed inference rules by logical operators in RDF Schemas. Moreover, the architecture of such a system, using Metalog and binary predicate calculus for querying RDF is not obvious. Finally, an implementation of this approach is still to be presented, in order to clarify the ideas of the authors in practice.

4. Conclusion

Studying these approaches, one can outline the requirements of a query language for RDF. The approaches that draw ideas from the knowledge representation and reasoning communities seem to be closer to the initial aims that led to the creation of RDF. However, it is obvious that such reasoning cannot be based on RDF. It would probably be more appropriate to define an abstract logical model for the query language, in terms of the RDF Graph Model. The first approach, which points to that direction, defining path expressions, has the drawback of being restricted to data queries, while it would be desirable to query the schema itself. One should also consider the Schema Discovery and Translation issue (as described in [9]): considering the discovery part trivial, there should be the

ability to understand RDF Schemas and derive information out of hierarchical relations and constraints, as well as information that is implied through inheritance.

5. Bibliography

[1] N. Sundaresan, RDF for XML, http://www.alphaworks.ibm.com

[2] A. Malhotra, N. Sundaresan , RDF Query Specification, *W3C Query Languages Workshop* (1998),

[3] R. V. Guha, O. Lassila, E. Miller, D. Brickley, Enabling Inferencing, *W3C Query Languages Workshop* (1998), http://purl.org/net/rdf/papers/QL98-enabling

[4] S. Decker, D. Brickley, J. Saarela, J. Angele, A Query and Inference Service for RDF, *W3C Query Languages Workshop* (1998), http://purl.org/net/rdf/papers/QL98-queryservice

[5] D. Brickley, S. Decker, J. Saarela, Classification Scheme Mapping – a simple demonstration using RDFIE and SiRPAC, http://purl.org/net/rdf/papers/classmap/

[6] M. Marchiori, J. Saarela, Query + Metadata + Logic = Metalog, http://www.w3.org/TandS/QL/QL98/pp/metalog

[7] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.P. Schnurr, R. Studer, A. Witt, On2broker: Improving Access to Information Sources at the WWW http://www.aifb.uni-karlsruhe.de/WBS/broker/inhalt-paper.html

[8] V. R. Benjamins, D. Fensel, Community is Knowledge in (KA)2, http://www.aifb.uni-karlsruhe.de/WBS/broker/inhalt-paper.html

[9] P. Valkenburg, D. Brickley, Query Language Issues in a Distributed Indexing Environment, *W3C Query Languages Workshop* (1998) http://purl.org/net/rdf/papers/QL98-distributed