SeRQL: A Second Generation RDF Query Language

Jeen Broekstra
Aduna
jeen.broekstra@aduna.biz

Arjohn Kampman
Aduna
arjohn.kampman@aduna.biz

4th November 2003

1 Introduction

RDF Query Language proposals are more numerous than fish in the sea[1], it seems. However, the most prominent proposals out there are query languages that were conceived as first generation tryouts of RDF querying, with little or no RDF-specific implementation and use experience to guide design, and based on an ever-changing set of syntactical and semantic specifications.

In this position paper, we introduce a set of general requirements for an RDF query language. This set is compiled from discussions between RDF implementors, our own experience and user feedback that we received on our work in Sesame [4], as well as general principles of query language design. We go on to show how we have compiled these requirements into drafting the SeRQL query language. SeRQL is explicitly *not* meant as 'yet another' query language: its aim is to reconcile ideas from existing proposals (most prominently RQL, RDQL and N3) into a proposal that satisfies a list of key requirements.

2 Requirements for RDF Querying

Alberto Reggiori and Andy Seaborne have collected a number of use cases and examples for RDF queries[2]. From this report, we can distill several general requirements for RDF queries. Apart from these requirements, several principles for query languages in general can be taken into account (see [1]), such as compositionality, and data model awareness.

From these sources and our experience in implementing and using first generation RDF query languages such as RQL [2] and RDQL [3], we have composed a list of key requirements for RDF Querying. In the next sections, we briefly discuss these requirements.

2.1 Compositionality

Compositionality is the principle that complex queries can be composed from smaller, simpler queries. In order to achieve this, it is imperative that the results returned by a query are in the same form as the original data. In other words: the output of one query can be used as input for the next query.

Perhaps surprisingly, many first generation RDF query language do not comply with this requirement: while the data model being queried is an RDF graph, the output of queries is often defined as a set of variable bindings or a table.

2.2 Schema Awareness

This requirement follows directly from the more general requirement of data model awareness: a query language should be aware of the structure it is querying. Whenever such structure is defined or inferred, a

[1] Thanks to #rdfig residents LotR, JibberJim and ndw for the expression
[2] see `http://rdfstore.sourceforge.net/2002/06/24/rdf-query/query-use-cases.html`
[3] see `http://www.hpl.hp.com/semweb/iswc2002/JenaTutorial.Alpha/RDQL/TutorialRDQL.html`

query language should be capable of exploiting this structure for type checking, optimization, inheritance, etc.

In the case of RDF, schema awareness translates directly to awareness of the semantics of RDF Schema primitives, such as class and property subsumption, domain and range contraints, etc. Most first generation RDF query languages solve this by doing inference separately from the query language and simply applying RDF queries to the entailed RDF graph. While this approach is certainly valid, the lack of schema-specific language constructs in the query language often make expressing schema-related queries awkward. Moreover, the fact that the semantics are not specified in the language specification itself introduces an ambiguity: different implementations of the same spec may result in different results on the same query, depending on how much of the RDF Semantics the implementation supports.

2.3 Optional Path Expressions

The nature of RDF as a framework for semistructured data requires that query languages are expressive enough to cope with data that is less rigidly structured than would be the case in, for example, a relational database. In particular, it often occurs in practice that for any given instance a particular property may or may not have a value. Query languages that are solely based on complete template matches fail to address such structures properly. A mechanism for expressing that a (part of a) match is *optional* is imperative.

2.4 Datatyping

Since the introduction of XML Schema datatypes [3] in RDF, support for inference and querying with respect to datatypes has become imperative. Most first generation query languages were defined prior to the introduction of datatypes and therefore lack language constructs for dealing with them.

3 Implementing the Requirements: SeRQL

As described in the introduction, SeRQL (Sesame Rdf Query Language, pronounced ”circle”) was developed as a second generation language.

A full user manual for SeRQL is available online[4]. In the next sections, we will illustrate how SeRQL implements the query language requirements identified in the previous sections, by means of several example queries. We use the museum dataset for our examples - this dataset is available online[5], where you can also find a SeRQL query engine in which the example queries can be evaluated.

3.1 Compositionality

In recognition of the fact that true compositionality requires the input and output data models to be compatible, SeRQL uses a query result mechanism that cannot only return not variable bindings, but also sets of RDF statements. This construction makes use of an alternative clause, called `CONSTRUCT`, that can be used as an alternative to the `SELECT` clause (which returns the familiar set of variable bindings).

The following example returns the set of all statements that have `http://www.european-history.com/picasso.html` as their subject.

```
CONSTRUCT
   *
FROM
   {<!http://www.european-history.com/picasso.html>} p {Y}
```

A more complex example shows how the construct query can also be used to *transform* rdf graphs, by defining a graph template in which variables that are bound in the `FROM` clause are re-used:

[4] see `http://sesame.aidministrator.nl/publications/SeRQLmanual.html`
[5] see `http://sesame.aidministrator.nl/sesame/serql/index.jsp?repository=museum`

```
CONSTRUCT
   {Painting} <my:createdBy> {Painter}
FROM
   {Painter} <rdf:type> {<cult:Painter>};
             <cult:creates> {Painting}
USING NAMESPACE
       cult = <!http://www.icom.com/schema.rdf#>,
       my = <!http://www.foo.com/bar#>
```

While the currently implemented SeRQL parser has no support (yet) for nesting SeRQL queries, it is evident that allowing graph tranformations and subgraph result sets is an important first step in allowing true compositionality.

3.2 Schema Awareness

The SeRQL engine in Sesame makes use of the fact that Sesame supports inferencing seperately from the query language. This approach is commonly taken in many query language implementations. However, SeRQL supports RDF Schema semantics in the language specification. While the implementation of the language in Sesame need not be concerned about these features, it is important to realize that the *specification* of the language explicitly defines these constructions as being schema-aware. Therefore, *any* implementation that supports SeRQL will have to implement Schema inferencing in order to be fully compatible.

While this increases the burden on the implementor, it makes life on users easier: since the specification of the language specifically defines that RDF Schema semantics are supported, any implementation of SeRQL will be required to return the same results.

SeRQL introduces a couple of schema-aware constructions that deserve special mention: `directSubClassOf`, `directSubPropertyOf` and `directInstanceOf`.

Definition 1 *A class X is a* direct subclass *of a class Y iff there is no class Z such that X is a subclass of Z and Z a subclass of Y (for X, Y and Z being distinct classes).*

As an example, the following query returns all subClassOf-statements that define *direct* subclasses of the class Artist.

```
CONSTRUCT
    *
FROM
   {Dsub} <serql:directSubClassOf> {<cult:Artist>}
USING NAMESPACE
      cult = <!http://www.icom.com/schema.rdf#>
```

Notice that the operation is supported using the existing syntax for path expressions, but that the result set will not contain the `directSubClassOf` predicate, since it is an operator, not a predicate occurring in the dataset.

3.3 Optional Path Expressions

As can be observed from the previous query examples, SeRQL uses a path expression syntax that closely follows the graph structure of RDF: it uses curly brackets for identifying nodes in the graph, while edges are denoted by predicate names or variables between nodes. A semicolon denotes a branching predicate from a preceding node, while a comma denotes a seperation between path expression (or, when used in a node, a pairwise disjoint multiple match for a subject or object).

An extra feature for supporting *optional path expressions* is the introduction of square brackets. By using this construction, we can specify parts of the path expression that are optional matches.

To illustrate, consider the following SeRQL query:

```
CONSTRUCT
   *
FROM
   {Artist} <rdf:type> {<cult:Artist>};
            <cult:first_name> {FName}
USING NAMESPACE
      cult = <!http://www.icom.com/schema.rdf#>
```

This query will return all Artists and their first names, but it will only return those artists for which a first name exists. To retrieve all Artists, even if they do not have a first name, we need to specify that the `first_name` part of the path is optional:

```
CONSTRUCT
   *
FROM
   {Artist} <rdf:type> {<cult:Artist>};
            [<cult:first_name> {FName}]
USING NAMESPACE
      cult = <!http://www.icom.com/schema.rdf#>
```

The above query will return all Artists, and if available also their first names.

3.4 Datatyping

SeRQL has explicit and implicit support for datatypes. The explicit support consists of an operator (`datatype()`) that can be used to access a literal's datatype. The URI returned by the operator can be used in a query just like any other URI.

The implicit support for datatypes concerns the supported comparison operators: $<$, $<=$, $=$, $!=$, $>$ and $>=$. The behaviour of the comparison operations between two literals depends on their datatypes. Two non-datatyped literals are only equal when their labels are identical, but two xsd:float-typed literals are equal when their numerical values are equal. So the literal `"100"^^<xsd:float>` is equal to both `"1.0e2"^^<xsd:float>` and `"100.00"^^<xsd:float>`.

The current implementation of the SeRQL query engine supports a subset of the XML Schema built-in datatypes, being xsd:boolean, xsd:float, xsd:double, xsd:decimal, and all subtypes of xsd:decimal.

4 Conclusion

SeRQL is an attempt to learn from past experience with implementing RDF query languages and to define a second generation query language that captures the structure and peculiarities of RDF and RDF Schema. The overview given here is by no means a complete overview of the SeRQL feature set. Instead, we have highlighted features that we consider important when looking at general requirements for RDF Query Languages.

We are looking for feedback and co-developers in further extending SeRQL, both as a proposal and as an implementation.

References

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.

[2] Sofia Alexaki, , V. Christophides, Gegory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb'01)*, pages 1–13, Hong Kong, May1 2001. See also `http://www.ics.forth.gr/RDF`.

[3] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. Recommendation, World Wide Web Consortium, May 2001. See `http://www.w3.org/TR/xmlschema-2/`.

[4] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Ian Horrocks and James Hendler, editors, *Proceedings of the first International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 54–68, Sardinia, Italy, June 9 – 12, 2002. Springer Verlag, Heidelberg Germany. See also `http://sesame.aidministrator.nl/`.

RDF Query Language proposals are numerous. However, the most prominent proposals are query languages that were conceived as first generation tryouts of RDF querying, with little or no RDF-specific implementation and use experience to guide design, and based on an ever changing set of syntactical and semantic specifications. In this chapter, we introduce a set of general requirements for an RDF query language.Â We go on to show how we have compiled these requirements into designing the SeRQL query language, and conclude that SeRQL can be considered a real second generation RDF querying and transformation language. Keywords. Query Language Graph Transformation Transformation Language Path Expression Formal Interpretation. A RDF query language is a computer language able to retrieve and manipulate data stored in Resource Description Framework format. SPARQL is emerging as the de-facto RDF query language, and is a W3C Recommendation. Released as a Candidate Recommendation in April 2006, it returned to Working Draft status in October 2006, due to open issues. It returned to Candidate Recommendation status in June 2007. On 12th November 2007 the status of SPARQL changed into Proposed Recommendation. On 15th January 2008 4 SeRQL: A Second Generation RDF Query Language 4.1 Introduction . . . 4.2 Query Language requirements . . . 4.2.1 Expressiveness and Adequacy . . . 4.2.2 Schema awareness . . .Â 2. A Query Language for RDF The Resource Description Framework (RDF) is the common foundational layer for representing machine processable information on the Web. We address re-search question B by specifying and implementing a query language for RDF, called SeRQL.Â 4. Inferencing Strategies for RDF The RDF specications include a formal semantics [Hayes, 2004] and a proof system that allows simple entailments.