A Comparison of RDF Query Languages
Kevin Hutt

ABSTRACT
This paper will present a comparison of current RDF-based query languages by putting forward query languages that represent a broad spectrum of architectural approaches to querying RDF-based data. Query languages reviewed in this paper include RDQL, ScRQL, SPARQL and XsRQL. A comparison of these languages will be done with respect to some of the more desired attributes of an RDF-based query language. Results of the comparisons will be presented which will provide insight into which language(s) are the most mature to date. Finally, we will conclude with some of the key challenges that lay ahead for any RDF-based query language to obtain wide spread adoption.

Keywords
RDF, RDFS, Query, Semantic Web

1. INTRODUCTION
The Resource Description Framework (RDF) is a flexible, extensible architecture to represent metadata information about World Wide Web resources [12]. As such, RDF provides a common framework for expressing web resource metadata so it can be automatically exchanged between computers without loss of meaning [9]. This framework will ultimately enable the machine processing of large amounts of metadata which will help to facilitate the evolution of the current World Wide Web into the next generation Semantic Web. With a Semantic Web, web users will no longer need to be an expert in the domain in which they seek resources in order to obtain the exact resource needed. Rather, web users will be able to utilize the metadata present in RDF-based web resources to enable more efficient information retrieval. Furthermore, by generalizing the concept of a web resource, RDF can be used to identify resources identified on the web, even though the resource may not be directly available on the web [9].

From a data model standpoint, RDF is based on identifying web resources (Subjects) utilizing Uniform Resource Identifiers (URI’s) and describing Subjects with their respective Properties and Property values. For instance, the RDF Statement “there is a person identified by a web resource whose full name is Kevin Hutt” can be represented by the following RDF graph.

The RDF Statement above is a directed graph which is composed of a Subject node, Property edge and Object node. An RDF Statement can be thought of as an assertion that a particular resource (Subject) exists with a specific Property and Object Value. Both the Subject and Object node are represented by a URI and an Object node can be represented by a URI or literal value. In this case, the Object is represented by a text literal.

Figure 1

The graph in Figure 1 can be also represented in an XML syntax which is the main syntax for exchanging RDF data.

1 <?xmlversion="1.0"?><rdf:RDF xmlns: rdf="http://www.w3.org/syntax-ns#" xmlns:exterms="http://www.rh.edu/vocab">
2  <rdf:Description rdf: about="http://www.rh.edu/~kevinh">
3    <exterms: fullname>Kevin Hutt</exterms: fullname>
4  </rdf:Description>
5  </rdf: RDF>

Line 1 is an XML declaration which indicates that the following content is XML. Line 2 indicates that the following XML content represents RDF. Line 3 declares another namespace which indicates that URI refs using the exterms prefix are associated with the vocabulary defined by the organization at http://www.rh.edu [9]. Line 4 starts with the Description element which is a container element for the rdf element being described. Line 4 also contains the about attribute which describes the subject of the statement. Line 5 describes the property and the property value (Object). Lines 6 and 7 are the closing tags for their respective elements [9].

There is currently no W3C recommendation for an RDF-based query language; however, a working group was formed in October 2004 with the goal of producing a candidate recommendation by April 2005. Prior to the W3C forming a working group, there have been many RDF-based query languages proposed and implemented over the last several years which have including implementations from industry heavyweights such as Hewlett Packard and Sun Microsystems. Most notably, Hewlett Packard submitted a W3C proposal in January 2004 for a query language called RDQL. A W3C standard for RDF query is an important milestone in the development of RDF-query languages as it will likely foster the necessary industry investment and facilitate widespread adoption. However, as might be expected with such an important technology, there exists a wide variety of proposed RDF-query languages each with its proponents. The open source community offers perhaps a different approach to standardization for RDF-query. Most notably the Sesame open source RDF database system has gained widespread acceptance and will likely be a major player with respect to standardization efforts. Standardization promises to be both a politically and technically challenging endeavor, but very important nonetheless.

Before presenting any query languages, it is first important to underscore some of the key motivations for a robust RDF-based query language:

* A declarative query language will provide easy access to RDF repositories which will enable programmers to deploy applications quickly and efficiently.
A good query language will leverage the semantics inherent in the RDF schemas and provide more meaningful results to the user.

A robust RDF-based query language will be able to work with many different schema languages since requiring each resource author to use the same schema language may not be possible [11].

An RDF-based query language is also needed in order to achieve the desired logical and physical data independence which allows the higher level language to specify the resources required and thereby letting the database engine determine how to store and access the resources [1].

The query paradigm allows for more intuitive data access with a shorter learning curve as well as making access possible in situations where there is higher operation overhead such as is the case with protocols such as SOAP [10].

2. QUERY LANGUAGE FEATURES
As aforementioned, there is no consensus about what an RDF-based query language should offer from a functionality standpoint. Although there exists a wide body of literature relating to expressivity within structured data models, expressivity relating to semi-structured models is still a relatively new research subject. However, even though many languages have been implemented with widely varying syntax, there exist many commonalities in the languages that have already been implemented with respect to overall functionality. For the purposes of this paper, a baseline set of query language features needs to be established to enable comparison between languages. This paper does not assert that the language features presented for a basis of comparison is an authoritative list, rather an informal list based on common desired features found in languages that range from the very simple to the more robust.

Before presenting a list of desired features for an RDF-based query language, it is important to note some of the peculiarities of the RDF data model which will ultimately impact the set of operations that should be provided by an RDF query language [4]. As stated previously, the underlying data model for RDF is a directed graph which is composed of a subject node and an object node which are connected by an edge which identifies the property. The combination of subject node, object node and property edge is commonly referred to as a triple which in essence makes an assertion about a given resource. As a result, the building blocks of queries are triple patterns which match the user-supplied triple patterns against the target graph and return a sub graph wherever the triple pattern finds a match [12]. The variables in the triple patterns, if any, are bound to the corresponding subject, predicate or object[12]. The simplest graph patterns are single triple patterns, but graph patterns can be combined using various graph operators into more complicated graph patterns which are essentially ‘anded’ together when queried against the target graph [12].

Whereas RDF allows the user to make simple statements about resources, RDF users will also want to define vocabularies used in those statements. Vocabularies allow users to define the kinds of resources (classes) and which properties are associated with those classes [9]. RDF Schema does not necessarily provide a vocabulary of application-specific classes, rather it provides facilities needed to describe classes and properties and also indicates which classes and properties are to be used together [9]. The ability to leverage the semantics of the RDF schema class associated with the RDF graph offers query writers powerful inference tools. And although application-specific vocabularies are not part of RDF/RDFS, much work has already been done in this area and vocabularies such as Dublin Core for web content authoring are beginning to gain widespread acceptance.

Finally, to further help orient the reader, many of the languages reviewed in this paper have a SQL-like syntax. Within SQL, the database is a closed world; the FROM clause identifies the tables in the database; the WHERE clause identifies constraints and can be extended with AND [10]. By analogy, within RDF query, the Web is the database and the FROM clause identifies the RDF resources available [10].

2.1 Basic Features
- **Generalized Path Expressions:** A path expression syntax for navigating RDF graph is required which matches a user-input graph pattern (subject-property-object) against the underlying graph model. Pattern matching features typically include:
  - Support for searching for a specific input pattern within the graph
  - Support for substituting variables in place of a node or property
  - Support for constraining values using Boolean expressions

- **Value comparison and Data Type Support:** Property values often contain literals which can be simple strings such as ‘joe jones’ or typed literals such as #12-27-04. An RDF-typed literal consists of a string literal and its URI reference (datatype URI’s) which are external to RDF [9].

Formally, a datatype is defined as a set of character strings known as the lexical space of d, a non-empty set called the value space of d and a mapping from the lexical space of d to the value space of d. This mapping is known as the lexical-to-value mapping of d [3]. For instance, let d be a Boolean datatype where the lexical space = {1,0} (i.e. the list of legal values to represent the data type) and the
value space = \{True, False\}. In this case, the lexical-to-value mapping maps the string 1 to True and the string 0 to False. In order to provide adequate support for value comparison, it is necessary for the query language to exploit any data types inherent in the RDF model.

- **Closure**: In the case of RDF, closure means that the results of any query operation are also graphs and not any other data structure. This means the results of the query can be used as input to another query and is useful when one wants to break a larger query into smaller queries[3].

**More Advanced Features:**

- **Semantic Capabilities**: One of the main advantages of the RDF data model is that it provides built-in support for semantic inference via RDFS. Semantic support of a query language might include:
  
  - The ability to query the subClass/SubProperties of the result set
  - The ability to query the subClass or SubProperties of the schema
  - The ability for a query language to support multiple schemas due to inability to force content providers to use just one schema [12]

- **Optional Values**: Most RDF-Query languages provide path expression support for exact matching on a graph. Some languages such as SPARQL provide support for specifying optional sections of the graph similar to an outer join in the world of SQL. With this functionality, results can be returned based on partial matches of pattern data.

- **Aggregate Functions**: Aggregate functions such as MIN, MAX and COUNT have long been supported in the structured world of SQL and could provide benefit to the RDF community. However, very few current languages support aggregation.

- **Advanced Set Operations**: Within the relational database model, several algebraic operations are implemented which can be combined. These 5 operations form the basis of relational completeness and include selection, projection, Cartesian product, set difference and set union [4]. The three basic operations selection, projection and product are supported by all of the languages presented in this paper via the path expression; however, some languages also support set union and set intersection [4].

3. QUERY LANGUAGE SURVEY

3.1 Overview

A survey of popular RDF query languages recently conducted by the W3C identified more than 20 languages that are either under development or have been implemented [6]. Some of the languages developed provide a basic level of pattern matching with no inference capabilities while others allow for more complex semantic processing. Still others were created as extensions of existing technologies languages such as X-Query to provide perhaps an easier transition for users of XML-based query languages. This paper will not perform a complete survey of all RDF-based query languages, but provide a sampling of languages that represent a broad cross-section of approaches including:

- A language with major mainstream industry support (RDQL) which gives insight into the level of current commitment within mainstream technology companies such as Hewlett Packard
- A language that has strong W3C support (SPARQL) which gives insight into the future direction of standardization efforts within the W3C
- A language that leverages current XML approaches (XsRQL) which gives insight into how RDF-Query can be added as an extension to other technologies
- A language with strong collaboration between the open source community and industry (SeRQL) and gives insight into an alternative approach to W3C standardization

In order to perform a simple side-by-side comparison of languages listed below, this paper will note whether each query language provides basic support for the 7 language features aforementioned. Whereupon a language offers advanced support for a feature listed below, it will be noted.

1. Value comparison and data type support
2. Generalized path expressions
3. Closure
4. Semantic capabilities
5. Optional values
6. Aggregate functions
7. Advanced set operations (union, intersection)

3.2 RDQL

RDQL was developed by Hewlett Packard and submitted to the W3C in January 2004. It has been implemented in several RDF systems including Jena, RDFStore, Sesame, PHP XML Classes, 3 Store and RAP-RDF API for PHP [13]. RDQL was derived mainly from SQUIBH, an earlier language. The syntax of RDQL is similar to a SQL-like select pattern where the select clause allows the projection of the variables [4]. Based on the limited amount of functionality offered in RDQL, it would appear that RDQL was designed to be a simple language. The RDQL approach “suggests a strategy for possible standardization: an RDQL-like language could be developed and deployed without detailed treatment of rule or inference facilities, yet subsequently we used to query ‘smarter’ RDF services which make use of inferences licensed by OWL or RDF-rule semantics” [15].

Syntax Example:

A simple path expression

```
SELECT ?x
WHERE (?x, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, http://example.com/someType >)
```

This select statement matches all statements in the graph that have a property of http://www.w3.org/1999/02/22-rdf-syntax-ns#type and object of http://example.com/someType. The variable ?x is bound to the subject resource [13].

Table 1: RDQL Scorecard

<table>
<thead>
<tr>
<th>Data Type Supp</th>
<th>Path Exp.</th>
<th>Closure Seman</th>
<th>Opt Values</th>
<th>Agg Funct</th>
<th>Adv Set Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

3.3 SPARQL

SPARQL has been designed to meet the requirements and design goals as described in the W3C RDF Data Access Working Group (DAWG) document “RDF Data Access Use Cases and Requirements” [12]. To date, SPARQL has not been endorsed by the working group and there have been a number of design issues raised to the working group. Although it is an incomplete implementation, it represents the latest work of the W3C and provides insight into the future direction of standardization efforts within the W3C. It would appear from the SPARQL Working Draft that the language developers are following an iterative development process vis-à-vis incorporating best practices from other languages as well as direct input from the W3C RDF Data Access Working Group. As such, it is difficult to say how much additional functionality SPARQL will have when it is finally complete.

SPARQL offers many of the basic features desired in an RDF-based query language. SPARQL provides a subset of operations on plain literals, XSD integers and XSD floats defined in XQuery and XPath functions and operators such as comparison of numeric values, functions on string values casting, comparison of duration, time and date values [12]. SPARQL treats an RDF graph as pure data and is not aware of any inference that an rdf store may provide [12]. One interesting feature of SPARQL is the RDF keyword CONSTRUCT where query results are returned as an RDF graph specified by an rdf template. Specifically, the CONSTRUCT keyword returns a single RDF graph formed as the union of the graph template with variable values obtained from each query result [12].

Syntax Example:

An example of how SPARQL implements Optional matching

```
PREFIX foaf: http://xmlns.com/foaf/0.1/
SELECT ?name ?mbox
WHERE ("x foaf:name ?name)
OPTIONAL (?x foaf:mbox ?mbox)
```

This query finds the names of people in the dataset and if there is a mbox property, it will return that as well. The main power of the OPTIONAL statement is that allows a partial match on the target graph similar to an outer join in SQL. Without the OPTIONAL keyword, each triple pattern in a WHERE clause must be satisfied because the triple patterns are essentially ‘anded’ together to enable the binding of variables to values.

Table 2: SPARQL Scorecard

<table>
<thead>
<tr>
<th>Data Type Supp</th>
<th>Path Exp.</th>
<th>Closure Seman</th>
<th>Opt Values</th>
<th>Agg Funct</th>
<th>Adv Set Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

3.3 SeRQL

SeRQL (pronounced “circle”) is loosely based on several earlier languages such as RQL, RDQL and N3 [4] and as such represents a second generation language. SeRQL’s aim is to reconcile ideas from existing proposals into a proposal that satisfies a lot of these key requirements [3]. The overall design goal of SeRQL includes compiling and implementing many of the best features from earlier languages and delivering a light-weight yet powerful language [4]. One interesting characteristic of SeRQL is that it
represents a collaboration between industry and the open source community and seemingly offers a viable alternative to RDF-query standardization over the W3C process. It is being developed as a collaboration between the Dutch software company Aduna and the open source community to address many of the concerns of the Sesame user community (www.openrdf.org) [3]. Sesame is an open source RDF database that offers support for many query languages including SeRQL which has become the default query language for Sesame.

SeRQL supports many of the basic RDF-query features such as path expressions, Boolean constraints and advanced features such as optional matching. Advanced features such as functions for aggregation are not supported.

Syntax Example:

One of the more interesting features of SeRQL is its support for advanced path expressions. In situations where one wants to query =two or more triples with identical subject and predicate, the subject and predicate do not have to be repeated.

SeRQL: [subj] pred1 [obj1, obj2, obj3]

This expression is equivalent to

[subj1] pred1[obj1]
[subj1] pred1[obj2]
[subj1] pred1[obj3] [3]

SeRQL also supports branched path expressions, which is useful when multiple properties result from a single node

SeRQL:[subj1] pred1[obj1]
            pred2[obj2]

which is equivalent to:

[subj1] pred1[obj1]
[subj1] pred2[obj2] [3]

Another interesting syntax example is how SeRQL supports RDF reification. Reification is where the subject or object of a statement is itself a statement. Since it is syntactic construction, it can be expressed using basic path expression syntax. [3]

[statement1] <rdf:type>[rdf:statement],
[statement1] <rdf:subject>[subj1],
[statement1] <rdf:predicate>[pred1],
[statement1] <rdf:object>[obj1],
[statement1] <rdf:pred2>[obj2],

This is a cumbersome way of dealing with reification. SeRQL allows one to treat reified statements with a shorthand.

SeRQL:[subj1] pred1 {obj1} pred2 {obj2} [3]

The principles of navigating path expressions are also applied to navigating class and property hierarchies within SeRQL [3]. To retrieve the subclasses of a particular class:<my:class1>

{subclass} <rdfs:subClassOf> <my:class1> [3]

The principles of navigating path expressions are also applied to navigating class and property hierarchies within SeRQL [3]. To retrieve the subclasses of a particular class:<my:class1>

{subclass} <rdfs:subClassOf> <my:class1> [3]

Table 3 SeRQL Scorecard

<table>
<thead>
<tr>
<th>Data Type Supp</th>
<th>Path Exp.</th>
<th>Closure</th>
<th>Sem Funct.</th>
<th>Opt Values</th>
<th>Agg Funct</th>
<th>Adv Set Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

3.4 XsRQL

XsRQL is an RDF query language that derives most of syntax and style from X-Query while leveraging many of the useful features developed by the W3C XML Query Working Group while omitting parts of the X-Query specification that are specific to XML and hence not required in an RDF environment [8]. The interaction between query evaluation and the data model in XsRQL is similar to the X-Query: as expressions are evaluated within XsRQL, they populate instances into the result set from the target graph. [8]. As the query processor evaluates higher expressions on the query tree, new items populate the result set or existing items disappear and the whatever remains at the top of the query tree are the results of the query [8]. Because of the relative maturity of XQuery, XsRQL really shines with its support of built in functions and currently supports string functions such as chr, empty, exists, string-length as well as aggregate functions such as count [8].

XsRQL does not follow the familiar SQL syntax of the language examples previously mentioned in this paper, rather it follows the slightly abbreviated syntax of XQuery.

Syntax Example

Give me every vcard :Full name in the repository

RDQL SQL-like syntax:

select ?x, ?fname where (?x <http://www.w3.org/2001/vcard-rdf/3.0#fn> ?fname) [8]

In the abbreviated world of XsRQL, the query would look like this:

@< http://www.w3.org/2001/vcard-rdf/3.0#fn/> [8]

Another example where we want to find within a datastore triples where the age of the first partner is less than that of the second partner

In RDQL:
SELECT ?x, ?y WHERE
(?x:marriedTo ?y)
(?x :age ?xAge)
(?y :age ?yAge)
and ?xAge < ?yAge

In XsRQL:
for $x in *[@<marriedTo>]
for $y in $x/@<marriedTo>/*
return
{$x,@marriedTo>, $y}[8]

This example more fully illustrates how different the XsRQL language syntax is compared to other RDF query languages.

Table 4 XsRQL Scorecard

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. RESULTS

Table 5 : Combined Scorecard

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Path Exp</th>
<th>Closur e</th>
<th>Seman Func.</th>
<th>Opt Values</th>
<th>Agg Funct.</th>
<th>Adv Set</th>
<th>Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeRQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SPARQL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RDQL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XsRQL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

5. CONCLUSIONS
From the basis of this scorecard, SeRQL provides the most language features to date with SPARQL coming in a close second. In addition, both SeRQL and SPARQL provide an easy to read SQL-like syntax. It would appear that SeRQL with strong support from the open source community and SPARQL with strong support from the W3C are leading the charge towards standardization. As previously stated, standardization for RDF Query is an important step in the evolution of the Semantic Web as it will likely foster the necessary widespread industry commitment to make the Semantic Web a success. It will be interesting to see how quickly the W3C can forward its standardization efforts with the open source community progressing as fast as it is. Languages such as XsRQL will likely struggle to gain a wide foothold due to the arcane syntax of the language. However, it is possible that one size may not fit all and one language may not be enough to support the diverse needs of the RDF community.

Some of the challenges to widespread adoption of RDF Query are tightly coupled with the challenges of the Semantic Web. As of January 2005, Peter Norvig, Director of Google’s Search Quality, reports that there are less than 200,000 documents on the web that have .rdf or .owl extension which is approximately .0005% of the web [16]. What is the point of building RDF query tools if there are no documents to query? [16]. Another challenge to the Semantic Web and consequently RDF Query is competing ontology’s or vocabularies [16]. Content providers within a well-defined vertical industry might be successful defining ontology’s based on earlier successes with ANSI X.12 which defines industry specific EDI transaction sets because customers often force suppliers to comply [16]. However, it will be difficult to enforce ontology’s within a broader pool of content providers. Lastly, RDF Query to provide value, the metadata people affix to content must be trustworthy and in some cases content providers may be motivated to provide inaccurate or misleading metadata to lure search users. In summary, their is a symbiotic relationship between RDF Query and the larger Semantic Web initiative and it will be important for RDF Query standardization efforts to align with the larger issues that the Semantic Web initiative is encountering in order to gain widespread adoption.

6. REFERENCES


21st Computer Science Seminar
SE1-T4-7