A Comparison of RDF Query Languages

Peter Haase[1], Jeen Broekstra[2], Andreas Eberhart[1], and Raphael Volz[1]

[1] Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany
{haase, eberhart, volz}@aifb.uni-karlsruhe.de
[2] Vrije Universiteit Amsterdam, The Netherlands
jbroeks@cs.vu.nl

Abstract. The purpose of this paper is to provide a rigorous comparison of six query languages for RDF. We outline and categorize features that any RDF query language should provide and compare the individual languages along these features. We describe several practical usage examples for RDF queries and conclude with a comparison of the expressiveness of the particular query languages. The use cases, sample data and queries for the respective languages are available on the web[6].

1 Introduction

The Resource Description Framework (RDF) is considered to be the most relevant standard for data representation and exchange on the Semantic Web. The recent recommendation of RDF has just completed a major clean up of the initial proposal [11] in terms of syntax [1], along with a clarification of the underlying data model [10], and its intended interpretation [7]. Several languages for querying RDF documents and have been proposed, some in the tradition of database query languages (i.e. SQL, OQL), others more closely inspired by rule languages. No standard for RDF query language has yet emerged, but the discussion is ongoing within both academic institutions, Semantic Web enthusiasts and the World Wide Web Consortium (W3C). The W3C recently chartered a working group[1] with a focus on accessing and querying RDF data. We present a comparison of six representative query languages for RDF, highlighting their common features and differences along general dimensions for query languages and particular requirements for RDF. Our comparison does not claim to be complete with respect to the coverage of all existing RDF query languages. However, we try to maintain an up-to-date version of our report with an extended set of languages on our website[6]. This extended set of languages also includes RxPath[2] and RDFQL[3].

Related Work. Two previous tool surveys [12,4] and diverse web sites[4] have collected and compared RDF query languages and their associated prototype

[1] http://www.w3.org/2001/sw/DataAccess/
[2] http://rx4rdf.liminalzone.org/RxPath
[3] http://www.intellidimension.com/
[4] http://www.w3.org/2001/11/13-RDF-Query-Rules/\#implementations

implementations. The web sites are usually focused on collecting syntactic example queries along several use cases. We follow this approach of illustrating a language but instantiate general categories of orthogonal language features to avoid the repetitiveness of use cases and capture a more extensive range of language features. Two tool surveys [12],[4] were published in 2002 and focused mainly only the individual prototype implementations comparing criteria like quality of documentation, robustness of implementation and to a minor extent the query language features[5] which changed tremendously in the past two years. We detail the feature set and illustrate supported features through example queries. [5] analyzes the foundational aspects of RDF data and query languages, including computational aspects of testing entailment and redundancy.

It should be stressed that our comparison does not involve performance figures, as the focus is on the RDF query languages, not the tools supporting these languages.

The paper is organized as follows. Section 2 elicits several dimensions that are important for designing and comparing RDF query languages. Section 3 introduces the six languages that are compared in this paper along with the implementations that we have used. Section 4 demonstrates RDF query use cases, which are grouped into several categories. These use cases are used to compare the individual languages and expose the set of features supported by each language. Section 5 presents a wish list for further important but yet unsupported query language features. We conclude in Section 6 with a summary of our results.

2 Language Dimensions

2.1 Support for RDF Data Model

The underlying data model directly influences the set of operations that should be provided by a query language. We therefore recapitulate the basic concepts of RDF and make note of their implications for the requirements on an RDF query language.

RDF abstract data model. The underlying structure of any RDF document is a collection of triples. This collection of triples is usually called the RDF graph. Each triple states a relationship (aka. edge, property) between two nodes (aka. resource) in the graph. This abstract data model is independent of a concrete serialization syntax. Therefore query languages usually do not provide features to query serialization-specific features, e.g. order of serialization.

Formal semantics and Inference. RDF has a formal semantics which provides a dependable basis for reasoning about the meaning of an RDF graph. This reasoning is usually called entailment. Entailment rules state which implicit information can be inferred from explicit information. Hence, RDF query languages can consider such entailment and might convey means to distinguish implicit from explicit data.

Support for XML schema data types. XML data types can be used to represent data values in RDF. XML Schema also provides an extensibility framework

[5] cf. [12][Table 2 on page 16] for the most extensive summary

suitable for defining new datatypes for use in RDF. Data types should therefore be supported in an RDF query language.
Free support for making statements about resources. In general, it is not assumed that complete information about any resource is available in the RDF query. A query language should be aware of this and should tolerate incomplete or contradicting information.

2.2 Query Language Properties

In addition to eliciting the support for the above RDF language features we will discuss the following properties for each language.

– *Expressiveness.* Expressiveness indicates how powerful queries can be formulated in a given language. Typically, a language should at least provide the means offered by relational algebra, i.e. be relationally complete. Usually, expressiveness is restricted to maintain other properties such as safety and to allow an efficient (and optimizable) execution of queries.
– *Closure.* The closure property requires that the results of an operation are again elements of the data model. This means that if a query language operates on the graph data model, the query results would again have to be graphs.
– *Adequacy.* A query language is called adequate if it uses all concepts of the underlying data model. This property therefore complements the closure property: For the closure, a query result must not be outside the data model, for adequacy the entire data model needs to be exploited.
– *Orthogonality.* The orthogonality of a query language requires that all operations may be used independently of the usage context.
– *Safety.* A query language is considered safe, if every query that is syntactically correct returns a finite set of results (on a finite data set). Typical concepts that cause query languages to be unsafe are recursion, negation and built-in functions.

3 Query Languages

This section briefly introduces the query languages and actual systems that were used in our comparison.

3.1 RQL

RQL [8] is a typed language following a functional approach, which supports generalized path expressions featuring variables on both nodes and edges of the RDF graph. RQL relies on a formal graph model that captures the RDF modeling primitives and permits the interpretation of superimposed resource descriptions by means of one or more schemas. The novelty of RQL lies in its ability to smoothly combine schema and data querying while exploiting the taxonomies of labels and multiple classification of resources. RQL follows an OQL-like syntax: `select Pub from {Pub} ns3:year {y} where y = "2004" using namespace ns3 = ...` RQL is orthogonal, but not closed, as queries return variable bindings instead of graphs. However, RQL's semantics is not completely compatible with the RDF Semantics: a number of additional restrictions are placed on RDF models to allow querying with RQL[6].

RQL is implemented in ICS-FORTH's RDF Suite[7], and an implementation of a subset of it is available in the Sesame system[8]. For our evaluation we used Sesame version 1.0, which was released on March 25, 2004.

3.2 SeRQL

SeRQL [3] stands for Sesame RDF Query Language and is a querying and tranformation language loosely based on several existing languages, most notably RQL, RDQL and N3. Its primary design goals are unification of best practices from query language and delivering a light-weight yet expressive query language for RDF that addresses practical concerns.

SeRQL syntax is similar to that of RQL though modifications have been made to make the language easier to parse. Like RQL, SeRQL is based on a formal interpretation of the RDF graph, but SeRQL's formal interpretation is based directly on the RDF Model Theory.

SeRQL supports generalized path expressions, boolean constraints and optional matching, as well two basic filters: select-from-where and construct-from-where. The first returns the familiar variable-binding/table result, the second returns a matching (optionally transformed) subgraph. As such, SeRQL construct-from-where-queries fulfill the closure and orthogonality property and thus allow composition of queries. SeRQL is not safe as it provides various recursive built-in functions.

SeRQL is implemented and available in the Sesame system, which we have used for our comparison in the version 1.0. A number of querying features are still missing from the current implementation. Most notable of these are functions for aggregation (minimum, maximum, average, count) and query nesting.

3.3 TRIPLE

The term Triple denotes both a query and rules language as well as the actual runtime system [16]. The language is derived from F-Logic [9]. RDF triples `(S,P,O)` are represented as F-Logic expressions `S[P->O]`, which can be nested. For example, the expression `S[P1->O1, P2->O2[P3->O3]]` corresponds to three RDF triples `(S,P1,O1)`, `(S,P2,O2)`, and `(O2,P3,O3)`.

[6] An example of such a restriction is that every property must have *exactly* one domain and range specified.

[7] `http://139.91.183.30:9090/RDF/`

[8] `http://www.openrdf.org/`

Triple does not distinguish between rules and queries, which are simply headless rules, where the results are bindings of free variables in the query. For example, `FORALL X <- ( X[rdfs:label->"foo"] )@default:ln.` returns all resources which have a label "foo". Since the output is a table of variables and possible bindings, Triple does not fulfill the closure property. Triple is not safe in the sense that it allows unsafe rules such as `FORALL X ( X[rdfs:label->"foo"] <- ( a[rdfs:label->"foo"] )@default:ln.`. While Triple is adequate and closed for its own data model, the mapping from RDF to Triple is not lossless. For example, anonymous RDF nodes are made explicit. Triple is able to deal with several RDF models simultaneously, which are identified via a suffix @model.

Triple does not encode a fixed RDF semantics. The desired semantics have to be specified as a set of rules along with the query. Datatypes are not supported by Triple. For the comparison, we used Triple in the latest version from March 14th, 2002 along with XSB 2.5 for Windows.

3.4 RDQL

RDQL currently has the status of a W3C submission [15].

The syntax of RDQL follows a SQL-like select pattern, where a from clause is omitted. For example, `select ?p where (?p, <rdfs:label>, "foo" )` collects all resources with label "foo" in the free variable p. The select clause at the beginning of the query allows projecting the variables. Namespace abbreviations can be defined in a query via a separate "using" clause. RDF Schema information is not interpreted. Since the output is a table of variables and possible bindings, RDQL does not fulfill the closure and orthogonality property. RDQL is safe and offers preliminary support for datatypes.

For the comparison, we worked with Jena 2.0 of August 2003.

3.5 N3

Notation3 (N3) provides a text-based syntax for RDF. Therefore the data model of N3 conforms to the RDF data model. Additionally, N3 allows to define rules, which are denoted using a special syntax, for example: `?y rdfs:label "foo" => ?y a :QueryResult` Such rules, whilst not a query language per se, can be used for the purpose of querying. For this purpose queries have to be stored as rules in a dedicated file, which is used in conjunction with the data. The CWM filter command allows to automatically select the data that is generated by rules. Even though N3 fulfills the orthogonality, closure and safety property, using N3 as a query language is cumbersome.

N3 is supported by two freely available systems, i.e. Euler [14] and CWM [2]. None of these systems do automatically adhere to the RDF semantics. The semantics has to be provided by custom rules. For our comparison, we worked with CWM in the version of March 21, 2004.

3.6 Versa

Versa takes an interesting approach in that the main building block of the language is a list of RDF resources. RDF triples play a role in the so-called traversal operations, which have the form `ListExpr - ListExpr -> BoolExpr`. These expressions return a list of all objects of matching triples. For instance, the traversal expression `all() - rdfs:label -> *` would return a list containing all labels. Within a traversal expression, we can alternatively select the subjects as well by placing a vertical bar at the beginning of the arrow symbol. Thus, `all() |- rdfs:label -> eq("foo")` would yield all resources having the label ”foo”. The fact that a traversal expression is again a list expression, allows us to nest expressions in order to create more complex queries.

The given data structures and expression tree make it hard to project several values at once. Versa uses the distribute operator to work around this limitation. It creates a list of lists, which allows selecting several properties of a given list of resources.

Versa offers some support for rules since it allows traversing predicates transitively. Custom built-ins, views, multiple models, and data manipulation are not implemented. However, Versa fulfills the orthogonality and safety criteria.

The Versa language is supported by 4Suite, which is a set of XML and RDF tools[9]. We used 4Suite version 1.0a3 for Windows from July 4, 2003 along with Python 2.3.

4 Use Cases

In this section we present use cases for the querying of RDF data and evaluate how the six query languages support them. In the following tables, “-” indicates no support, “•” full support and “◦” partial support.

4.1 Sample Data

For our comparison, we have used a sample data set[10]. It describes a simple scenario of the computer science research domain, modelling persons, publications and a small topic hierarchy. The data set covers the main features of the RDF data model. It includes a class hierarchy with multiple inheritance, data types, resources with multiple instantiations, reification, collections of resources, etc. These variety of features are exploited in the following use cases.

4.2 Use Case Graph

Due to RDF’s graph-based nature, a central feature of most query languages is the support for graph matching.

[9] `http://www.4suite.org`
[10] Available at
`http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/sample.rdf`

Table 1. Path expression query in all languages

N3	Triple
{ ?y s:author ?z. ?z ?p ?x. ?x a s:Person; s:name ?result. } => { ?result a :QueryResult.}.	FORALL C,X,A,N <- (s:Paper[s:author->C] AND C[X->A] AND A[s:name->N])@rdfschema(s:ln).
RDQL	Versa
SELECT ?n WHERE <s:Paper>, <s:author>, ?c), (?c, ?collection, ?a), (?a, <s:name>, ?n) USING s FOR ...	QUERY=((s:Paper - s:author -> *) - properties(.) -> *) - s:name -> *
SeRQL	RQL

select PersonName
from {X} <s:author> {} <rdfs:member>
{} <s:name> {PersonName}
using namespace
s = <!...>
	select PersonName
	from {X} s:author {y}. rdfs:member
	{z}. s:name {PersonName}
	using namespace
	s = ...
	rdfs = ...

The namespace **s** is bound to that of the sample data.

Path Expressions. The central feature used to achieve this matching of graphs is a so-called path expression, which is typically used to traverse a graph. A path expression can be decomposed into several joins and is often implemented by joins. It comes at no surprise that path expressions are offered - in various syntactic forms (cf. Table 1) - by all RDF query languages.

Return the names of the authors of publication X

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Path	●	●	●	●	●	●

Optional Path Expressions. The RDF graph represents a semi-structured data model. Its weak structure allows to represent irregular and incomplete information. Therefore RDF query languages should provide means to deal with irregularities and incomplete information. A particular irregularity, which has to be accounted for in the following query, is that a given value may or may not be present.

What are the name and, if known, the e-mail of the authors of all available publications ?

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Optional Path	-	-	●	●	-	○

Unfortunately, only two languages - namely Versa and SeRQL - provide built-in means for dealing with incomplete information. For example, the SeRQL language provides so-called optional path expressions (denoted by square brackets) to match paths whose presence is irregular:

```
SELECT PersonName, Email FROM
{X} <ns3:author> {} <rdfs:member>
{p} <ns3:name> {PersonName};
[<ns3:email> {Email}]
USING NAMESPACE
   ns3 = <!...>
```

Usually, such optional path expressions can be simulated, if a language provides set union and negation. A correct answer is provided by unifying the results of two select-queries, where the first argument of the union retrieves all persons with an e-mail address, the second those without an e-mail address. Consequently, RQL gets partial credit since these operations are supported. Another workaround is possible in rule languages like N3 and Triple by defining a rule, which infers a dummy value in absence of other data. In our example, an optional path would carry the dummy email address. Since this workaround might create undesired results if the absence of values is checked in other parts of the query, we do not credit the respective languages. Versa's distribute operator allows formulating the query by converting the list of authors into a list of lists of (optional) attributes.

4.3 Use Case Relational

RDF is frequently used to model relational structures. In fact, n-ary tables such as found in the the relational data model can easily be encoded in RDF triple.

Basic algebraic operations. In the relational data model several basic algebraic operations are considered, i.e. (i) selection, (ii) projection, (iii) cartesian product, (iv) set difference and (v) set union. These operations can be combined to express other operations such as set intersection, several forms of joins, etc. The importance of these operations is accounted by the definition of relational completeness. In the relational world, a given query languages is known to be relationally complete, if it supports the full range of basic algebraic operations mentioned above.

The three basic algebraic operations selection, projection, product are supported by all languages and have been used in the path expression query of the previous use case *Graph*. We therefore concentrate on the other two basic operations mentioned above, i.e. union and difference, in this section.

Union. As we have seen in the previous section, union is provided by RQL. Versa contains an explicit union operator as well. N3 and Triple can simulate union with rules.

Return the labels of all topics that and (union) the titles of all publications

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Union	-	●	-	●	●	●

Difference. Difference is a special form of negation:

Return the labels of all topics that are not titles of publications

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Difference	-	-	-	○	-	●

While difference is described in the Versa documentation (but not implemented) RQL also provides an implementation of this algebraic operator.

The following RQL query provides the correct answer:

```
( select title
  from s:Topic{T}. rdfs:label {title}
) minus (
  select title
  from s:Publication{P}. s:title {title} )
using namespace
  s = ... , rdfs = ...
```

Quantification. An existential predicate over a set of resources is satisfied if at least one of the values satisfies the predicate. Analogously, a universal predicate is satisfied if all the values satisfy the predicate. As any selection predicate implicitly has existential bindings, we here consider universal quantification.

Return the persons who are authors of all publications.

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Universal	-	-	-	-	-	●

RQL is the only language providing the universal quantification needed to answer this query:

```
SELECT person
FROM s:Person{person}
WHERE FORALL z IN
    (SELECT x FROM s:Publication{x} )
  SUCH THAT EXISTS p IN
    (SELECT Y FROM {z} s:author {}. rdfs:member {y})
  SUCH THAT person = p
USING NAMESPACE s = ..., rdfs = ...
```

4.4 Use Case Aggregation and Grouping

Aggregate functions compute a scalar value from a multi-set of values. These functions are regularly needed to count a number of values. For example, they are needed to identify the minimum or maximum of a set of values. Grouping additionally allows aggregates to be computed on groups of values.

Aggregation. A special case of aggregation tested in the following query is a simple count of the number of elements in a set:

Count the number of authors of a publication.

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Counting	-	-	-	●	●	●

Counting is supported by N3, Versa and RQL. The following N3 rule gives the appropriate answer:

```
{?y.sam:author math:memberCount ?result .} =>
{:Query :Result ?result}.
```

Grouping. None of the compared query languages allows to group values, such as provided with the SQL GROUP BY clause[11].

4.5 Use Case Recursion

Recursive queries often appear in information systems, typically if the underlying relationship is transitive in nature. Note that the RDF query engine must handle *schema* recursion, i.e. the transitivity of the subClassOf relation. The scope of this use case is *data* recursion introduced by the application domain. In the sample datasettopics are defined along with their subtopics, where the subtopic property is transitive. This must be expressed in the query.

Return all subtopics of topic "Information Systems", recursively.

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Recursion	-	●	-	●	●	-

Triple (and N3), being rule-based systems, naturally can support the required recursion through the definition of auxiliary rules:

```
FORALL O,P,V  O[acm:SubTopic->V] <-
  EXISTS W  (O[acm:SubTopic->W] AND W[acm:SubTopic->V])@default:ln.
FORALL Y <-
    ('...#ACMTopic/':Information_Systems[acm:SubTopic->Y])@default:ln.
```

Versa does not support general recursion, but provides a keyword "traverse", which effects a transitive interpretation of a specified property. This suffices to answer our recursive query:

```
traverse(@"...#ACMTopic/Information_Systems",
    acm:SubTopic, vtrav:forward, vtrav:transitive )
```

[11] The RQL query language allows the computation of global aggregate values such as required for a query such as selecting the publication with the maximum number of authors.

4.6 Use Case Reification

Reification is a unique feature of RDF. It adds a meta-layer to the graph and allows treating RDF statements as resources themselves, such that statements can be made about statements. In the sample data reification is used to state who entered publication data. Hence, the following query is of interest:

Return the person who has classified the publication X.

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Reification	○	○	●	○	-	○

SeRQL and Triple support reification with a special syntax. In SeRQL, a path expression representing a single statement can be written between the curly brackets of a node:

```
select Person
from {{X} <s:isAbout> {}} <dc:creator> {Person}
using namespace s = <!...>
```

In Triple statements can be reified by placing them in angle brackets and using this within another statement: `FORALL V,W,X,Y,Z <- ( V[W-><X[Y->Z]>] )`. However, we were only able to use this feature in the native F-Logic syntax, since the reified statements in the RDF sample data were not parsed correctly.

While N3 cannot syntactically represent RDF reification, RDQL, RQL and Versa treat reified statements as nodes in the graph, which can be addressed through normal path expressions by relying on the RDF normalization of reification. This allows treating the reification use case like any other query. We show the Versa example below:

```
QUERY=(all() |- rdf:predicate -> s:isAbout) - dc:creator -> *
```

4.7 Use Case Collections and Containers

RDF allows to define groups of entities using collections (a closed group of entities) and containers, namely *Bag*, *Sequence* and *Container*, which provide an intended meaning of the elements in the container. A query language should be able to retrieve the individual and all elements of these collections and containers, along with order information, if applicable, as in the following query:

Return the last author of Publication X

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Sequences	○	○	○	○	○	○

Although none of the query languages provide explicit support for the processing of containers (as known for example from the processing of sequences in XQuery), in all query languages it is possible to query for a particular element in a container with the help of the special predicate <rdf:_n>, which allows to address the *n*th element in a container. However, this approach only allows to retrieve the last element of a container if its size is known before.

None of the query languages provide explicit support for ordering or sorting of elements, except for Versa which features a special sort operator. RQL does have specific operators for retrieval of container elements according to its specification, but this feature is not implemented in the current engine.

4.8 Use Case Namespaces

Namespaces are an integral part of any query language for web-based data. The various examples presented so far showed how the languages allow introducing namespace abbreviations in order to keep the queries concise. This use case evaluates which operations are possible on the namespaces themselves. Given a set of resources, it might be interesting to query all values of properties from a certain namespace or a namespace with a certain pattern. The following query addresses this issue. Pattern matching on namespaces is particularly useful for versioned RDF data, as many versioning schemes rely on the namespace to encode version information.

Return all resources whose namespace starts with "http://www.aifb.uni-karlsruhe.de/".

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Namespace	○	-	●	-	●	●

SeRQL, RQL and N3 allow for pattern matching predicates on URIs in the same manner as for literals, which allows to realize the query as shown in the following for N3:

```
{?a ?b ?c. ?a log:rawUri ?d.
 ?d string:startsWith "http://www.aifb.uni-karlsruhe.de/" } =>
{:Query :Result ?d}.
```

For RDQL, the string match operator is defined in the grammar, however the implementation is incomplete. Versa has a contains operator, which apparently only works for string literals, not URIs.

4.9 Use Case Language

RDF allows to use XML-style language tagging. The XML tag enclosing an RDF literal can optionally carry an xml:lang attribute. The respective value identified the language used in the text of the literal. Possible values include en for english or de for german. This use case examines, whether the various languages support this RDF feature.

Return the German label of the topic whose English label is "Database Management"

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Language	-	-	●	-	-	-

Out of the compared languages, SeRQL is the only one that has explicit support to query language specific information. SeRQL provides a special function to retrieve the language information from a literal:

```
select deLabel
from {} <rdfs:label> {deLabel, enLabel}
where lang(deLabel) = "de" and lang(enLabel) = "en" and
      label(enLabel) = "Database Management"
```

4.10 Use Case Literals and Datatypes

Literals are used to identify values such as numbers and dates by means of a lexical representation. In addition to plain literals, RDF supports the type system of XML Schema to create typed literals. An RDF query language should support the XML Schema datatypes. A datatype consists of a lexical space, a value space and lexical to value mapping. This distinction should also be supported by an RDF query language. The sample data for example contains a typed integer literal to represent the page number of a publication, with the following two queries we will query both the lexical space and the value space:

Return all publications where the page number is the lexical value '08'

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Lexical Space	●	●	●	●	●	●

Return all publications where the page number is the integer value 8

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Value Space	○	-	●	-	-	●

All query languages are able to query the lexical space, but most query languages have no or only preliminary support for datatypes and do not support the distinction between lexical and value space. RDQL and SeRQL provide support for datatypes using a special syntax to indicate the datatype. However, the RDQL query did not work correctly in the implementation.

4.11 Use Case Entailment

RDF Schema vocabulary supports the entailment of implicit information. Two typical use cases in the context of RDF are:

– Subsumptions between classes and properties that are not explicitly stated in the RDF Schema,

– Classification of resources: For a resource having a property for which we know its domain – or analogously the range – is restricted to a certain class, we can infer the resource to be an instance of that class.

With the following query we evaluate the support of the query languages for RDF Schema entailment. The query is expected to return not only the resources for which the class membership is provided explicitly, but also those whose class membership can be inferred based on the entailment rules. Obviously, several other queries would be necessary to test the full RDF-S support. Due to space limitations, we restrict ourselves to the following query:

Return all instances of that are members of the class Publication

Query	RDQL	Triple	SeRQL	Versa	N3	RQL
Entailment	○	○	●	-	○	●

Discussion. Regarding the support for RDF-S entailment, the query languages take different approaches: RQL and SeRQL support entailment natively and even allow to distinguish between subclasses and direct subclasses. RDQL leaves entailment completely up to the implementation of the query engine, it is thus not part of the semantics of RDQL. Nevertheless, it gets partial credit, since Jena provides an optional mechanism for attaching an RDF–S reasoner such that the query processor takes advantage of it. N3 and Triple require an axiomatization of the RDF-S semantics, i.e. a set of rules. Versa provides no support.

The following sample shows how the query is realized in RQL:

```
select publications
from ns3:Publication{publications}
using namespace
  ns3 = <!http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/sample.rdf#>
```

5 Wish List

In the previous sections we have seen that the currently available query languages for RDF support a wide variety of operations. However, several important features - listed in the following text - are not well supported, or even not supported at all.

Grouping and Aggregation. Many of the existing proposals support very little functionality for grouping and aggregation. Functions such as min, max, average and count provide important tools for analysing data.

Sorting. Perhaps surprisingly, except for Versa, no language is capable to do sorting and ordering on the output. Related to this seems to be that many query languages do not support handling of ordered collections.

Optional matching. Due to the semi-structured nature of RDF, support for optional matches is crucial in any RDF query language and should be supported with a dedicated syntax.

Adequacy. Overall, the languages' support for RDF-specific features like containers, collections, XML Schema datatypes, language tags, and reification is quite poor. Since these are features of the data model, the degree of adequacy among the languages is low. For instance, it would be desirable for a query language to support operators on XML Schema datatypes, as defined in [13], as built-ins.

6 Conclusion

We believe that defining a suitable RDF query language should be a top priority in terms of standardization. Querying is a very fundamental functionality required in almost any Semantic Web application. Judging from the impact of SQL to the database community, standardization will definitely help the adoption of RDF query engines, make the development of applications a lot easier, and will thus help the Semantic Web in general.

We have evaluated six query language proposals with quite different approaches, goals, and philosophies. Consequently, it is very hard to compare the different proposals and come up with a ranking. From our analysis, we identify a small set of key criteria, which differ vastly between the languages.

A key distinction is the support for RDF Schema semantics. Languages like N3 and Triple do not make a strict distinction between queries and rules. Thus, a logic program representing the desired semantics, in this case RDF-S, can optionally supplement a query. SeRQL and RQL support RDF-S semantics internally. Versa takes a pragmatic approach by supporting the transitive closure as a special operator. While this is not very flexible, it solves most of the problems like traversing a concept hierarchy. RDQL's point of view is that entailment should be completely up to the RDF repository.

Orthogonality is a very desirable features, since it allows combining a set of simple operators into powerful constructs. Out of the six candidates, RQL, SeRQL, N3, and Versa support this property. Versa uses sets of resources as the basic data structure, whereas RQL, N3 and SeRQL operate on graphs. Triple can mimic orthogonality via rules, whereas RDQL does not support it.

Furthermore, we consider the extent to which the various use cases are supported. Obviously, one would have to distinguish between features like the support for recursive queries that fundamentally cannot be expressed in a language and a feature that simply has not been implemented in the respective system like a simple string match operator. However, since this distinction is often hard to make, we simply add up the queries that could be expressed. If the query could be formulated with a workaround, we count half a point. Using this metric, RQL and SeRQL appear to be the most complete languages, covering 10.5 and 8.5 out of 14 use case queries. Versa and N3 follow with 7.5 and 7. Triple and RDQL were able to answer the least queries and got 5.5 and 4.5 points.

Finally, we consider the readability and usability of a language. Obviously, this depends very much on personal taste. Syntactically, RQL, RDQL, and SeRQL are very similar due to their SQL / OQL heritage. Triple and N3 share the rules character. The Triple syntax allows for some nice syntactic variants. Versa's style is quite different, since a query directly exposes the operator tree.

Acknowledgments. Research reported in this paper has been partially financed by the EU in the project IST-2003-506826 SEKT.

References

1. D. Beckett. RDF/XML Syntax Specification (Revised). W3C Working Draft, 2003. Internet: `http://www.w3.org/TR/rdf-syntax/`.
2. T. Berners-Lee. CWM - closed world machine. Internet: `http://www.w3.org/2000/10/swap/doc/cwm.html`, 2000.
3. Jeen Broekstra and Arjohn Kampman. SeRQL: An RDF Query and Transformation Language. To be published, 2004. `http://www.cs.vu.nl/~jbroeks/papers/SeRQL.pdf`.
4. D. Fensel and A. Perez. A survey on ontology tools. Technical Report OntoWeb Deliverable 1.3, OntoWeb consortium, May 2002. http://www.ontoweb.org/ download/deliverables/D13_v1-0.zip.
5. Claudio Gutiérrez, Carlos A. Hurtado, and Alberto O. Mendelzon. Foundations of semantic web databases. In *Proceedings of the Twenty-third Symposium on Principles of Database Systems (PODS), June 14-16, 2004, Paris, France*, pages 95–106, 2004.
6. Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A comparison of rdf query languages. Technical report, University of Karlsruhe, 2004. `http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/`.
7. Patrick Hayes. Rdf semantics. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/\#rules`.
8. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Schol. RQL: A Declarative Query Language for RDF. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, Honolulu, Hawaii, USA, May7-11 2002.
9. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.
10. G. Klyne and J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Data Model. W3C Working Draft, 2003. Internet: `http://www.w3.org/TR/rdf-concepts/`.
11. O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Working Draft, 1999. Internet: `http://www.w3.org/TR/REC-rdf-syntax/`.
12. A. Maganaraki, G. Karvounarakis, V. Christophides, D. Plexousakis, and T. Anh. Ontology storage and querying. Technical Report 308, Foundation for Research and Technology Hellas, Institute of Computer Science, Information Systems Laboratory, April 2002.
13. Ashok Malhotra, Jim Melton, and Norman Walsh. Xquery 1.0 and xpath 2.0 functions and operators, w3c working draft 12 november 2003. `http://www.w3.org/TR/xpath-functions/`.
14. J. De Roo. Euler proof mechanism. Internet: `http://www.agfa.com/w3c/euler/`, 2002.
15. Andy Seaborne. Rdql - a query language for rdf, w3c member submission 9 january 2004, 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
16. M. Sintek and S. Decker. TRIPLE - an RDF query, inference and transformation language. In *Deductive Databases and Knowledge Management (DDLP)*, 2001.