Querying the RDF: Small Case Study in the Bicycle Sale Domain

Ondřej Šváb, Vojtěch Svátek, Martin Kavalec, and Martin Labský

Department of Information and Knowledge Engineering,
University of Economics, Prague, W. Churchill Sq. 4, 130 67 Praha 3, Czech Republic
{xsvao06,svatek,kavalec,labsky}@vse.cz

Abstract. We examine the suitability of RDF, RDF Schema (as simple ontology language), and RDF repository *Sesame*, for providing the back-end to a prospective domain-specific web search tool, targeted at the offer of bicycles and their components. Actual data for the RDF repository are to be extracted by analysis modules of a distributed knowledge-based system named *Rainbow*. Attention is paid to the comparison of different query languages and to the design of application-specific templates.

1 Introduction

The goal of *semantic web* initiative is to endow web data with formal syntax and semantics and thus make them available for automated reasoning. Such reasoning could improve information retrieval (moving from keyword-based to content-based retrieval), but also increase the degree of automation in data/application integration or website design. Among the 'semantic web' representation languages, RDF[1] has prominent position. It is used to express interconnected logical facts, which can be *semantically* viewed as simple kind of structured *knowledge*. On the other hand, semantic web facts may potentially arise in large quantities, and thus, at the *syntactical level*, require treatment similar to traditional, tabular *data*. Several tools have been developed for RDF storage and retrieval, such as *Jena*[2], *RDF Suite* [1], and finally *Sesame*, which is the focus of this paper. There are also multiple query languages implemented in different tools.

Although semantic annotations written by hand have the best quality, it is unrealistic to obtain the critical mass of semantic web purely manually. Automated annotation of legacy pages (or service descriptions) by means of lingustic or statistical techniques became a hot topic in semantic web research[3]. In this paper, we analyse the applicability of RDF query languages and of the *Sesame* repository for storage and retrieval of facts potentially discovered by *Rainbow*—a distributed knowledge-based system for analysis of web content and structure.

[1] http://www.w3.org/RDF

[2] http://www.hpl.hp.com/semweb/jena2.htm

[3] Cf. the workshop on 'Human Language Technology for Semantic Web and Web Services' at the last International Semantic Web Conference, http://gate.ac.uk/conferences/iswc2003.

© V. Snášel, J. Pokorný, K. Richta (Eds.): Dateso 2004, pp. 84–95, ISBN 80-248-0457-3.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2004.

Section 2 of this paper discusses how facts (on bicycle sales) extracted by *Rainbow* can be represented in RDF and shows the underlying RDF Schema. Section 3 describes the architecture of *Sesame*. Section 4 compares the major RDF query languages with respect to the given application. Finally, section 5 deals with application-specific templates that can shield a user of the (prospective) semantic search tool from the syntax of query languages.

2 Representing the *Rainbow* Results in RDF

2.1 Architecture of *Rainbow*

The loosely-coupled architecture of *Rainbow* consists of a *web spider*, a *full-text database tool*, and several *analytical tools*, interfacing with each other by means of *web service* protocols. Analytical tools specialise in different forms of web data, such as free text, text structured with HTML tags, website topologies or images. The semantics of services is modelled by an *application ontology*. In the current state of the system, the services can only be invoked procedurally, in a fixed order; the ontology is hence merely used indirectly, at design time of the composed application. A more flexible composition solution (based on skeletal planning) is envisaged for the future. More details can be found in [8] and at the project homepage[4]. The current application of *Rainbow* aims at development of a semantic search tool for the domain of *bicycle products*. Websites of bicycle-selling companies[5] are systematically analysed, with emphasis on catalogue information, but also including a general profile of the company.

2.2 Resource Description Framework and Vocabulary Language

Resource description framework[6] (RDF) is a language developed for representing information about resources in the World Wide Web; it can however be used as general language for encoding facts. RDF statements, also called *triples*, consist of three parts: *subject*, *predicate* (property), and *object*. An statement may e.g. say that a particular web page was created by a particular human: the page then is the subject, the human is the object, and the relation 'created by' is the predicate. Any real-world entity (and even property) can be understood as resource, whether accessible via the web or not. Object is the only part of statement where not only a resource but also a *literal* (i.e. simple string or numerical value) may appear. RDF literals can also be *typed*, via reference to *XML Schema datatypes*. Even whole statements can be declared as resources: this technique is called *reification*[7], and enables to assert facts about statements themselves. To identify a particular information resources, RDF uses *URI*, the

[4] `http://rainbow.vse.cz`

[5] As initial data source, we use the sites referenced by the *Google Directory*, namely its node `Sports/Cycling/BikeShops/Europe/UK/England`.

[6] `http://www.w3.org/RDF`

[7] From Latin: *res*= 'thing', since the statement thus becomes an object of discourse.

Uniform Resource Identifier. RDF statements may be encoded using varying *serialization syntax*[8] but always conform to the same, *graph-oriented data model*, where subjects and objects are represented by nodes, predicates by directed arcs, and each node-arc-node triple represents one RDF statement.

RDF is endowed with more expressive power through RDFS: the *RDF Schema* [2], which plays the role of vocabulary language. Individual resources can be assigned types, i.e. *classes*; RDFS then allows to build a *hierarchical structure* over classes and properties, and to declare the *domain* and *range* of properties.

2.3 RDF Facts and RDFS Ontology on Bicycle Sites

When applied on bicycle-selling sites, analytical modules of Rainbow are typically able to extract the *name* of a bike, its *price*, details on its *components* (such as fork, frame, rear derailer etc.), its *picture*, and possibly some information about the *company* that offers it. Bikes, as well as separately-sold bike components are associated with *retail offers*. Examples of information 'triples' (in free-text form, to avoid syntax issues) are "Company X offers bike Y". "Bike Y has name Rockmachine Tsunami", "Bike Y has fork Z". "Fork Z has name Marzocchi Air", "Price of bike Y is 2500." Furthermore, we need to represent *metadata* associated with the extracted facts, such as "Statement XY has certainty 0.75" or "Statement XY was produced by URL analysis module".

The *RDF schema* (i.e. simple ontology) of our domain uses four namespaces: `bike` dealing with bikes themselves, `comp` dealing with (not necessarily 'bike') companies, `pict` dealing with pictures on web pages, and `meta` dealing with metadata on statements extracted by *Rainbow*. Its graph is shown on Fig. 1 and 2 (decomposed for easier readability). The central point of the schema is the concept of RetailOffer. It corresponds to an offer of BikeProduct (whole bike or component) by a Company; it is also associated with the Name under which and Price for which it is offered, and URL of associated Picture. URI of particular RetailOffer corresponds to the URL of catalogue item containing the offer[9]. BikeProduct is superclass of all bike products. Note that BikeProduct and its subclasses only have 'types' of products as their instances, not individual physical entities. Such 'type' of product can be offered for different prices and even under slightly different names (associated with the given instance of RetailOffer) and accompanied with different pictures, while BikeProduct itself has a 'canonical' name, specified e.g. by its manufacturer. Finally, let us explain the nature of *metadata*. Our solution to representing them is based on reification and inspired by the SWAP project[10]. In order to store metadata about e.g. origin, confidence,

[8] The standard format is *RDF/XML* (see `http://www.w3.org/RDF`); there is also a line-based encoding format, *N-triples* (see `http://www.w3.org/TR/2002/WD-rdf-testcases-20020429`) and a format easily readable for humans, *Notation 3* (N3, see `http://www.w3.org/2000/10/swap/Primer`).

[9] Typically the place from where the core information was extracted.

[10] Ongoing IST project on Semantic Web and Peer-to-Peer (knowledge nodes), see `http://swap.semanticweb.org/`

security and caching of each piece of knowledge, they set up a complex metadata schema [3]. In contrast, we are only need a few metadata items, such as, information on which *analysis module* the statement was obtained from, or its *certainty factor*. Metadata are grouped under an abstract class called Meta.

3 Architecture of *Sesame*

Sesame allows persistent storage and querying of RDF data and schema[11]; it consists of three functional modules:

- *Query Module* parses a query, builds the query tree to optimise it, and finally evaluates it in a streaming fashion.
- *Admin Module* enables to insert and delete RDF data and schema, checks for consistency of newly added statements with statements in the repository, and infers entailed information. Inferencing is done according to rules and axioms defined in [6] as well as to custom (application-specific) rules/axioms[12].
- *Export Module* exports the content of repository (data or schema or both).

Additionally, *Sesame* uses a stack of SAILs (Storage and Inference Layers), which transparently ensure access to specific implementations of repository. The underlying repositories can be based on a (relational or object-oriented) DBMS, RDF files cached in memory, RDF network services or existing RDF stores.

Our choice of *Sesame* was to some extent motivated by our close contacts with its developers. There are at least two comparable tools for RDF storage and retrieval. *RDF Suite*[13] [1], developed by ICS FORTH, Greece, is potentially faster for large queries thanks to flexible adaptation of database schema to the given RDF schema. It supports the full RQL query language (see section 4.1), and enables dynamic loading of multiple RDF schemata. *Jena*[14], developed by HP Labs, Bristol, UK, offers a user-friendly interface for writing RDF schemata, and an API for ontology languages (OWL, DAML+OIL, RDFS). It only supports the RDQL query language (see section 4.3). Arguments in favour of *Sesame* might be close adherence to the most recent 'inference-centric' updates of RDF, and some features of its original query language, SeRQL (see section 4.2). Given the experimental nature of our project, response time, reliability (typically decreasing with increasing role of inference), and quality of editing interface do not play so crucial a role, and since we deal with a single RDF Schema fully under our control, there is no need for dynamic schema integration. In our setting for *Sesame*, we further opted for RDBMS back-end.To enable easy search in the bicycle data repository, an *HTML interface* is being developed, with pre-fabricated query templates (cf. section 5).

[11] The first stable version, *Sesame 1.0RC1*, can be downloaded from `http://sourceforge.net/projects/sesame`. *Sesame* is developed by the Dutch company *Aduna* (earlier *Aidministrator*), see `http://sesame.aidministrator.nl`.

[12] In our case, it is e.g. possible to define a rule stating that property partOfModel is inverse to property hasBikePart.

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

[14] `http://www.hpl.hp.com/semweb/jena2.htm`

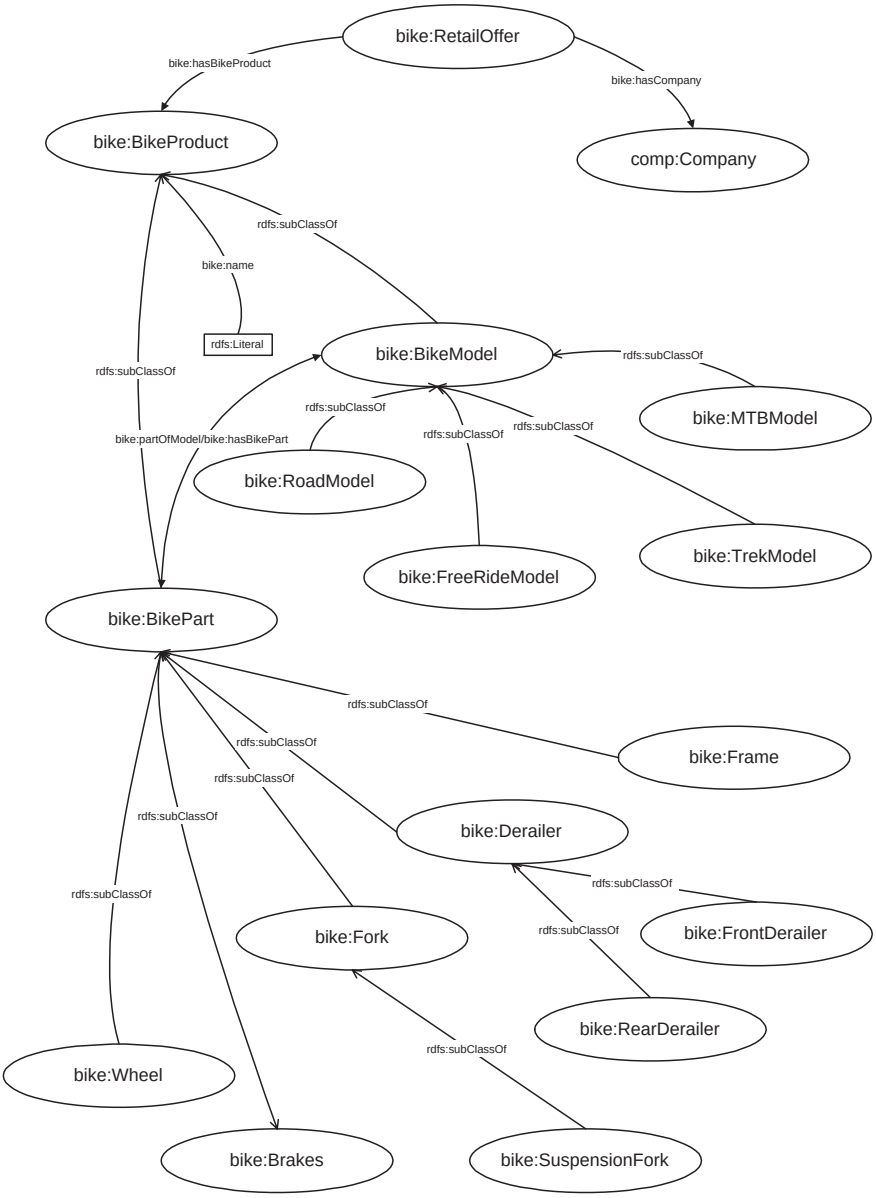

Fig. 1. RDF schema of bicycle domain 1/2

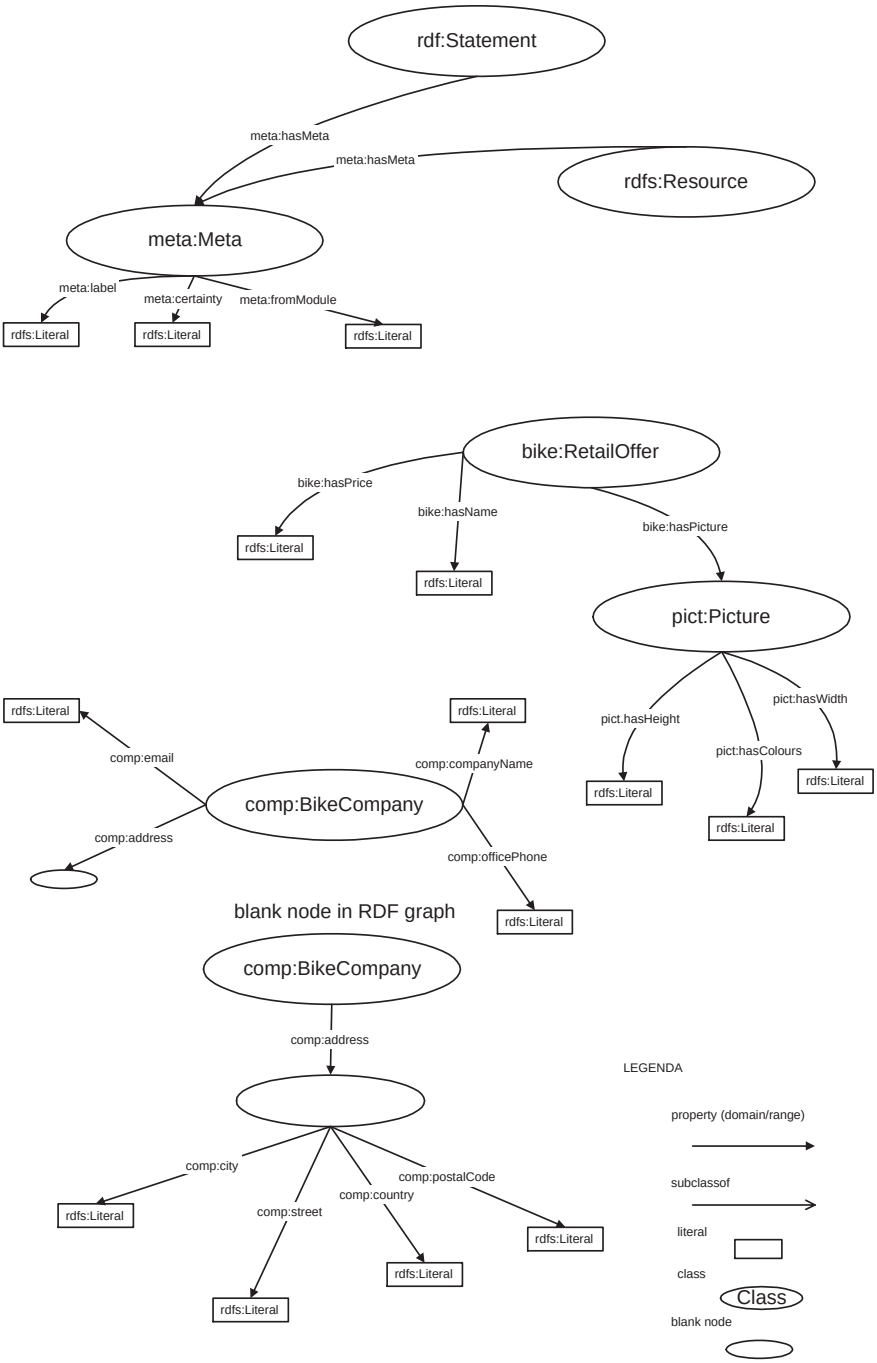

Fig. 2. RDF schema of bicycle domain 2/2

4 Querying the RDF in *Sesame*

There are three RDF query languages implemented in *Sesame*: RQL, RDQL and SeRQL. Unlike standard SQL, these languages only serve for querying and not for data manipulation. The main accent is laid on RQL and SeRQL, which enable querying both the RDF data and associated RDF schemata. We will demonstrate RQL and SeRQL on examples, and attempt to expose their weak and strong aspects. We will also briefly mention RDQL and the language implemented in the PerlRDF tool produced by Ginger Alliance (i.e. not supported by *Sesame*).

4.1 RQL

RQL is a declarative query language over RDF and RDFS. It was originally proposed in the context of *RDF Suite* [1]; its implementation in *Sesame* is only partial and adheres more closely to the recent RDF updates by the W3C. The building blocks of RQL are *functions* and *path expressions*. Functions enable to directly query the RDF Schema. Examples of functions are `Class` (returning the set of all classes), `Property` (returning the set of all properties), `domain` or `range` (returning the domain/range of a property). They can be be combined in many ways, even with path expressions.

RQL offers the SELECT-FROM-WHERE construct known from SQL, with some differences. It has two obligatory and two optional parts. In the (obligatory) SELECT clause, we list the variables (selected from the subsequent FROM clause), the values of which we want in the result. We could also use an asterisk, representing all variables. In the (obligatory) FROM part, we specify the RDF subgraph over which we query, in the form of path expression representing a filter on the graph. Further conditions are expressed is the (optional) WHERE clause. A WHERE expression is typically a *comparison* of variables from FROM clause and concrete values; we can also compare a variable with the result of an embedded query (see example 2Q), and use boolean *connectives*. RQL offers the comparison operators `<`,`>`,`=`,`>=`,`<=` and `like` (string matching, with possible left or right expansion). The RQL query engine tries to convert the operands to be compared to the same type. It is doing in this sequence: classes, properties (both compared hierarchically), real numbers, integers, literals and finally, resources. The last clause is (optional) USING NAMESPACE; it enables to write elements from certain namespace in short form (prefix:local name = Qname)[15].

Now we demostrate the use of RQL on examples related to our bicycle application; they conform to the RDF Schemata in Fig. 1 and 2. Path expressions of the queries are shown in Fig. 3.

Let us first demonstrate the use of RQL functions.

1Q: *Find restrictions (domain and range) of property hasWidth.*

```
select domain(@predicate), @predicate, range(@predicate)
from  {} @predicate {}
where @predicate = pict:hasWidth
```

[15] In all examples in this paper we omit namespace definitions, for brevity.

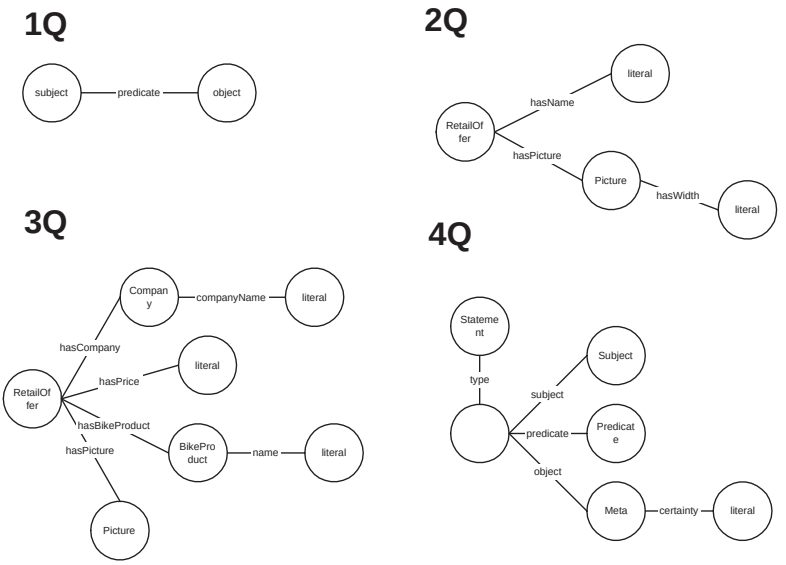

Fig. 3. Path expressions for sample queries

The next example already demonstrates the use of path expressions.
2Q: *Find all retail offers with name starting with letter "l" and having a picture with width lower than 70.*

```
select *
from {X : bike:RetailOffer } bike:hasName {name},
     {X} bike:hasPicture {Y}. pict:hasWidth {width}
where name like "l*" and width < 70
```

We see that in RQL, a variable (denoting a resource) can be assigned class (here, RetailOffer) using *shortcut notation* (with colon). This query also features lexicographic and numerical *comparisons* and a slightly more complicated *path expression*. It contains two paths starting from the same node specified by variable X. In the node specified by variable Y, the path is extended across the arc (property) hasWidth (see also Fig. 3).

RQL is however not very suitable for expressing *optional* path expressions, as manifested on the following example (note the last sentence).
3Q: *Find all retail offers of bicycles that have a concrete bike component. Output the name of company that offers the bike, the picture of retail offer, the price of bike (offer). Retrieve the retail offer even if the URL of picture is not known.*

In *RQL* must be this type of query expressed by applying a Boolean union on the results of two partial queries; also notice the use of operator `in` (reference to embedded query) in the second subquery, in order to eliminate duplicate results (obviously, with increasing number of optional parts of the query graph we would face combinatorial explosion):

```
(select web, company, price, picture, name
 from {X : bike:RetailOffer } bike:hasCompany
      {web : comp:Company }.  comp:companyName {company},
      {X} bike:hasPrice {price},
      {X} bike:hasPicture {picture},
      {X} bike:hasBikeProduct {idtyp}. bike:name {name}
 where  idtyp=data:part1)
union
(select web, company, price, null, name
 from {X : bike:RetailOffer } bike:hasCompany
      {web : comp:Company }.  comp:companyName {company},
      {X} bike:hasPrice {price},
      {X} bike:hasBikeProduct {idtyp}. bike:name {name}
 where  idtyp=data:part1
        and not (X in select X
                      from {X} bike:hasPicture {picture}))
```

4.2 SeRQL

SeRQL [4] ("*Sesame* RDF Query Language", pronounced as 'circle') is a declarative query language over RDF and RDF Schema; in contrast to RQL, there is explicit support for *optional path expressions*. There are two alternative types of queries, SELECT and CONSTRUCT. While SELECT returns a table of results, CONSTRUCT returns again an RDF graph, which is part of the graph being queried or derived from it via introducing new properties or classes. The parts of SELECT have the same meaning as in RQL; the only difference of CONSTRUCT is in the CONSTRUCT clause itself, where a structure of RDF triples appears in the place of variable list. An analogy of RQL functions are SeRQL *built-in predicates*, they however only cover some of the queries that could not be made using the RDF representation of the schemata themselves. This is the case of `<serql:directSubClassOf>`, `<serql:directSubPropertyOf>` and `<serql:directType>`, since the information on e.g. direct vs. inferred subclass relationship is not preserved in the repository in RDF form. In contrast to RQL, *XML datatypes* can be used in queries. SeRQL does not have an implicite sequence of type comparisons: unless the RDF literals themselves are typed (contain information on their datatypes), we must explicitly say how the query engine should compare two expressions in the WHERE clause, e.g. `WHERE width < "150"^^<xsd:positiveInteger>`. In SeRQL, we can, again, use the Boolean connectives AND, NOT and OR (in WHERE clause). In the following, we will translate the examples (1Q, 2Q and 3Q) to SeRQL.

1Q: *Find restrictions (domain and range) of property hasWidth.*
There are no built-in construct for accessing the domain/range. They can however be retrieved in the *RDF representation of the schema*, which is a part of the repository. We also benefit from *shortcut notation*: multiple edges from the same node are separated with semicolon, and the node is not repeated any more:

```
select domain, range
from {&lt;pict:hasWidth&gt;} &lt;rdfs:domain&gt; {domain}; &lt;rdfs:range&gt; {range}
```

2Q: *Find all retail offers that have a name starting with letter "l" and their picture has width lower than 70.*

```
select *
from {X} <bike:hasName> {name},
     {X} <bike:hasPicture> {Y} <pict:hasWidth> {width}
where name like "l*" and width < "70"^^<xsd:integer>
```

3Q: *Find all retail offers of bicycles that have a concrete bike component. Output the name of company that offers the bike, the picture of retail offer, the price of bike (offer). Retrieve the retail offer even if the URL of picture is not known.*

```
select prv, web, company, price, picture, name
from {prv} <serql:directType> {<bike:RetailOffer>};
           <bike:hasPrice> {price};
          [<bike:hasPicture> {picture}];
           <bike:hasBikeProduct> {idtyp},
     {idtyp} <bike:name> {name},
     {prv} <bike:hasCompany> {web} <rdf:type> {<comp:Company>};
                                   <comp:companyName> {company}
where idtyp = <data:part1>
```

The query shows a strong aspect of SeRQL: *optional path expressions* (in brackets). Also notice the combination of shortcut and not-shortcut notations.

The last, new example (we omitted its RQL form for brevity) deals with *reified* statements (with the abstract 'meta' resource, see section 2).
4Q: *Find all statements that have certainty higher than 0.9.*
Queries to reified statements may use their own *shortcut form* in SeRQL:

```
select *
from { {reifSubj} reifPred {reifObj} }
         <meta:hasMeta> {obj} <meta:certainty> {certainty}
where certainty > "0.9"^^<xsd:double>
```

We choose SeRQL for our application, mainly because of the need for *optional path expressions* (since we deal with often incomplete data extracted from HTML pages) and shortcut querying of *reified statements*. The strong aspect of RQL—functions for direct querying of RDF Schema—was found idle for our purpose, since we deal with relatively small and stable schemata.

4.3 Comparison with Other Languages

RDQL [7] was originally developed for the *Jena* tool. Its version in *Sesame* takes into account RDF data with schema but without inferential capability. It allows to specify a path expression but without support for optional parts. The SELECT clause has different syntax but usual meaning. There is no FROM clause, and the graph pattern is specified in the WHERE clause, as list of triples;

partial paths are bound together with variables. Finally, the AND clause specifies filters on variable values and the USING clause maps to namespaces.

Now we demonstrate RDQL on one example: *find all retail offers which have not the name* `liberta` *and their picture has width lower than 70.*

```
SELECT ?retailoffer, ?name, ?picture, ?width
WHERE (?retailoffer, <rdf:type>, <bike:RetailOffer>) ,
      (?retailoffer, <bike:hasName>, ?name) ,
      (?retailoffer, <bike:hasPicture>, ?picture) ,
      (?picture, <pict:hasWidth>, ?width)
AND ( ?width < 70 && ?name ne "liberta")
```

The last query language we mention is part of *PerlRDF*, a collection of tools developed by Ginger Alliance (`http://www.gingerall.com`). It offers path expressions, comparisons, functions and namespaces; there is no support for inferencing nor optional path expressions. We illustrate this RDF query language on the query: *find all retail offers that have a picture with width more than 10.*

```
Select ?retailoffer, ?name,
       ?picture->[http://rainbow.vse.cz/schema/picture.rdfs#hasHeight],
       ?picture->[http://rainbow.vse.cz/schema/picture.rdfs#hasWidth]
From bike:RetailOffer::?retailoffer->bike:hasName{?name},
            ?retailoffer->bike:hasPicture{?picture}
Where
  ?picture->[http://rainbow.vse.cz/schema/picture.rdfs#hasWidth] > '10'
```

5 Query Templates for the Bicycle Application

In order to make our prospective RDF repository available for a casual user, we decided to prepare a domain-specific *HTML interface* with several (SeRQL) *query templates*. The templates should *shield* the user from the syntax of the query language, and even offer a very simple form of *navigational retrieval*. Our idea is based on two-stage querying. The template for *initial query* (specifying its FROM part) is quite complicated, rich in optional path expressions:

```
from {idretail} <rdf:type> {<bike:RetailOffer>};
               [<bike:hasCompany> {idweb}
                     <rdf:type> {<comp:Company>};
                     <comp:companyName> {company};
                     <comp:address> {} <comp:city> {city}];
               [<bike:hasPrice> {price}];
               [<bike:hasPicture> {picture}];
                <bike:hasBikeProduct> {idbike}
                     <rdf:type> {<bike:BikeModel>};
                     <bike:name> {name}
                    [<bike:hasBikePart> {idFork}
                          <rdf:type> {<bike:Fork>};
                          <bike:name> {Fork}];
```

The final shape of the query will be tuned by the user, who may refine the SELECT clause (variables), FROM clause (optional or not), and WHERE clause (comparisons). The results of the initial query are the starting point for *follow-up querying*. For example, the initial query might be: *Find all bikes that are sold by this company*. The results contain various information about each retail offer of the company. There might be e.g. the fact that some offered product has a certain type of frame. Now the user can choose (i.e. click on) the option 'offer', which means: *Query on all retail offers of this product* (i.e., this type of frame). Follow-up queries will be mediated by simpler templates such as for pictures, compaines, retail offers of a particular type of bike or component and so on.

6 Conclusions and Future Work

We discussed the way a concrete RDF storage-and-retrieval tool, *Sesame*, can be used for our specific application within the *Rainbow* project, presented the *RDF schema* for this application, and analysed the *RDF query languages* implemented in *Sesame* Eventually, SeRQL was found suitable for our purposes. We plan to experimentally evaluate our hypotheses on a repository filled with a solid amount of *real-world data*. The data will be accessible through a *domain-specific HTML interface* with pre-fabricated *query templates* as well as through queries manually compiled by the user. In conjuction with this first live experiment, we will also have to *integrate* the results of multiple analytical modules of *Rainbow*.

The project is partially supported by grant no. 201/03/1318 of the Grant Agency of the Czech Republic.

References

1. Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Tolle K.:The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, 2^{nd}International Workshop on the Semantic Web, in conjunction with WWW10, Hongkong, 2001.
2. Brickley D., Guha R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World-Wide Web Consortium, Feb. 2004
3. Broekstra J., Ehrig M., Haase P., van Harmelen F., Kampman A., Sabou M., Siebes R., Staab S., Stuckenschmidt H., Tempich C.: A Metadata Model for Semantics-Based Peer-to-Peer Systems. In: Proceedings of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing, Budapest, 2003.
4. Broekstra J., Kampman A.: User Guide for Sesame. `http://sesame.aidministrator.nl/publications/users/`
5. Broekstra J., Kampman A.: Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema, On-To-Knowledge project deliverable 10, 2001.
6. Hayes P., McBride B.: RDF Semantics. W3C Recommendation, World-Wide Web Consortium, Feb. 2004, `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.
7. Seaborne A.:A Programmer's Introduction to RDQL, `http://jena.sourceforge.net/tutorial/RDQL`, April 2002
8. Svátek V., Kosek J., Labský M., Bráza J., Kavalec M., Vacura M., Vávra V., Snášel V.: Rainbow - Multiway Semantic Semantic Analysis of Websites. In: 2^{nd} DEXA Int'l Workshop on Web Semantics, Prague, IEEE Computer Society Press 2003.