Generating Application Ontologies from Reference Ontologies

Marianne Shaw[1], MS, Landon T. Detwiler[2], MS,
James F. Brinkley[1,2,3], MD, PhD, and Dan Suciu[1], PhD
Departments of Computer Science and Engineering[1], Biological Structure[2],
and Medical Education and Biomedical Informatics[3]
University of Washington, Seattle, WA

Abstract

The semantic web provides the possiblity of linking together large numbers of biomedical ontologies. Unfortunately, many of the biomedical ontologies that have been developed are domain-specific and do not share a common structure that will allow them to be easily combined. Reference ontologies provide the necessary ontological framework for linking together these smaller, specialized ontologies.

We present extensions to the semantic web query language SparQL that will allow researchers to develop application ontologies that are derived from reference ontologies. We have modified the ARQ query processor to support subqueries, recursive subqueries, and Skolem functions for node creation. We demonstrate the utility of these extensions by deriving an application ontology from the Foundational Model of Anatomy.

Introduction

In recent years there has been a proliferation of ontologies for biomedical information. These ontologies are being developed for use in specific subdomains and are not necessarily structured to be part of a larger ontological framework. Although the semantic web provides the ability to combine these ontologies, the lack of cohesive structure across them can limit meaningful combinations.

Reference ontologies [1] have been proposed as a mechanism for providing the necessary ontological framework for linking together these smaller, domain-specific ontologies. Reference ontologies are specifically intended to cover a broad range of related information. As a result, they often contain more information than needed by specialists working in particular biological subdomains.

Techniques are needed to generate application ontologies from these larger reference ontologies. Simply getting a slice of a reference ontology is not sufficient. These techniques must enable scientists to combine information from one or more reference ontologies, as well as permit the addition or modification of information as appropriate for their specific subdomain.

Many biological ontologies have been intentionally developed so that they can become part of the semantic web. These ontologies have been either created or exported into RDF [2] or OWL (Web Ontology Language) [3]. SparQL is the semantic web query language recommended by the W3C [4] for querying RDF; several SparQL implementations are publicly available. For these reasons, we have chosen to extend SparQL over other semantic web query languages.

In this work we present vSparQL, an extension of the SparQL syntax that enables the creation of application ontologies as views of reference ontologies. Specifically, our extension adds support for subqueries, recursive queries, and Skolem functions for the creation of new nodes. We demonstrate the utility of our extensions through the development of an application ontology from a reference ontology – we generate a radiologist view over the Foundational Model of Anatomy reference ontology.

Motivation

The Foundational Model of Anatomy (FMA) [5] is a reference ontology that represents the structure of the human body using a combination of classes and relationships. The FMA is developed in the Protege [6] frame-based system and has been exported to OWL. It contains more than 75,000 classes representing structures in the body, 120,000 terms associated with these classes, and 168 different types of relationships. Classes represent structures as small as cellular components and as large as the body itself. The model contains over 2.1 million relationships.

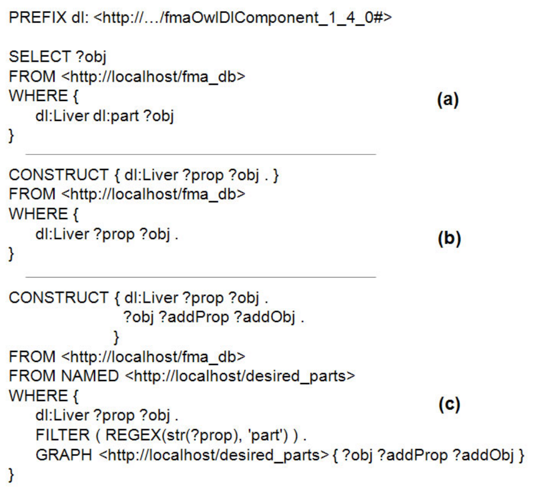
The FMA contains too much information for researchers interested in developing their own application ontologies. Specialists want a portion of the FMA that corresponds to their subdomain; this may involve a small subset of the classes, terms, and relationships available. Additionally, they may want to augment or modify the information from the FMA.

Consider an application ontology that might be derived from the FMA by a radiologist looking at images of the human liver. The radiologist is interested in the parts of the liver that are large enough to be seen in his images. All organs that are not part of the liver need not be included in his ontology; cellular and sub-cellular components can be ignored. He is only interested in the relationship that indicates which elements of the liver are parts of each other, but he also wants to keep the structure provided by the FMA to ensure that his ontology is ontologically sound. For example, he wants to retain the hierarchical "is_a" information which indicates that the liver is a lobular organ, which is itself a parenchymatous organ, which is a solid organ, etc. However, the FMA structure may be too refined for his purposes and some elements may need to be removed. For example, "Organ" is composed of "Cavitated organ" and "Solid organ;" the radiologist may collapse these subclasses back into "Organ."

Our long-term goal is to enable application ontologies to be constructed as "views" through a series of queries to reference ontologies. Queries can be developed to extract the portions of the reference ontology that are desired for application ontologies; additional queries can then be applied to modify or augment these extracted portions. In this model, a researcher can query his application ontology by executing his new queries over the non-materialized view; the query results would be obtained by querying the original reference ontology.

In this paper we present a query language to enable this model of creation for application ontologies. Queries must be able to extract and return portions of an ontology without extensive knowledge of the reference ontology itself. We would like to extract portions of the ontology by recursively following properties within the ontology. Queries must be composable; the results of subqueries should be valid input to subsequent queries. Additionally, it must be possible to combine results from queries over different ontologies. Finally, queries must be able to construct new nodes based upon the results of subqueries.

Query languages have been developed for the semantic web, including RQL [7], SparQL [8], and Triple [9]. View languages (RVL [10], Triple views [11], [12]) have been developed for specifying views over RDF ontologies; these view languages do not provide for generalized recursive queries.



```
PREFIX dl: <http://…/fmaOwlDlComponent_1_4_0#>

SELECT ?obj
FROM <http://localhost/fma_db>
WHERE {                                                    (a)
    dl:Liver dl:part ?obj
}
------------------------------------------------------------
CONSTRUCT { dl:Liver ?prop ?obj . }
FROM <http://localhost/fma_db>
WHERE {
    dl:Liver ?prop ?obj .                                  (b)
}
------------------------------------------------------------
CONSTRUCT { dl:Liver ?prop ?obj .
                  ?obj ?addProp ?addObj .
                }
FROM <http://localhost/fma_db>
FROM NAMED <http://localhost/desired_parts>
WHERE {                                                    (c)
    dl:Liver ?prop ?obj .
    FILTER ( REGEX(str(?prop), 'part') ) .
    GRAPH <http://localhost/desired_parts> { ?obj ?addProp ?addObj }
}
```

Figure 1: **SparQL queries.**

SparQLer [13], PSPARQL [14], and GLEEN [15] provide queries with the ability to recursively follow paths to match on. Noy [16] does provide the ability to extract portions of an ontology by following relationships; however, this work has been implemented in the Protege environment and is not readily composable with other queries. Schenk [17] adds view-like subqueries to SparQL; we have leveraged this approach and added the ability to have recursive subqueries.

We have chosen to extend SparQL to provide the necessary functionality for deriving application ontologies from reference ontologies. We introduce basic SparQL syntax in the next section before presenting our SparQL extensions.

SparQL

SparQL is a semantic web inspired query language designed for querying Resource Description Framework (RDF) models. RDF models are built up from a series of triples. Triples are of the form (subject, predicate, object) where subjects are resources, predicates are properties, and objects are values. SparQL queries consist of triple patterns that are evaluated against an underlying RDF graph to find matches. We use examples to illustrate properties of the query language.

Figure 1(a) presents a basic SparQL SELECT query. This query returns all of the parts of the liver. In this query, PREFIX defines a namespace to be used within this query; one can use the shorthand "dl:" to replace the full string throughout the query. SELECT indicates what should be returned by the query. In this example, ?obj is a variable that will be bound when the query triple pattern matches the model. The FROM keyword indicates the data source over which the query should be run. WHERE contains a set of triple patterns that are matched against the underlying model.

Figure 1(b) contains a CONSTRUCT query. Instead of simply returning a list of specified values, CONSTRUCT queries return RDF graphs using the results of the matching WHERE clauses. This query returns an RDF graph containing all of the direct properties of the liver.

Several advanced features of SparQL can be seen in Figure 1(c). This query returns all of the parts of the liver found in http://localhost/fma_db that are also found in http://localhost/desired_parts. The FROM keyword causes the associated RDF graph to be put into the default graph that a SparQL query is matched upon. Alternatively, RDF graphs specified using FROM NAMED can be queried through the use of the GRAPH keyword in WHERE clauses. The GRAPH keyword is followed by the name of the RDF graph to query (http://localhost/desired_parts) and a set of triples that should be matched against only that graph (?obj ?addProp ?addObj). FILTER clauses put additional constraints on the triples that match the underlying graph. In this example, the property matching ?prop must contain the string 'part' (e.g. part, regional_part, attributed_part).

vSparQL

Our extension to the SparQL query language is called vSparQL. The vSparQL extension enables the creation of application ontologies from reference ontologies. It does this by providing support for subqueries, recursive subqueries, and constructing new nodes.

vSparQL provides support for subqueries by allowing the data source specified by FROM or FROM NAMED to be the result of a CONSTRUCT query. This is possible because CONSTRUCT queries return RDF graphs; the results of a CONSTRUCT query can be used as a data source in a subsequent query. Any number of subqueries can be chained together using this technique.The subquery syntax is:

```
FROM <data source> [ CONSTRUCT  ...]
FROM NAMED <data source> [ CONSTRUCT ...]
```

Unmodified SparQL can be used to generate the same results as those returned by subqueries. However, subqueries provide for query modularity and reuse, potentially eliminating redundant computation. Additionally, vSparQL's subqueries make composing queries trivial; to apply a new

```
PREFIX dl: <http://.../fmaOwlDlComponent_1_4_0#>
PREFIX lp: <http://.../liverOnt>

CONSTRUCT { ?s ?p ?o .
             ?o ?addProp ?addObj .
           }
FROM NAMED <subquery> [
   CONSTRUCT { dl:Liver ?prop ?obj }
   FROM <http://localhost/fma_db>                          (a)
   WHERE { dl:Liver ?prop ?obj .
              FILTER ( REGEX(str(?prop), 'part') ) .
   }
] # subquery
FROM NAMED <http://localhost/desired_parts>
WHERE { GRAPH <subquery> { ?s ?p ?o } .
          GRAPH <http://localhost/desired_parts> { ?o ?addProp ?addObj . }
}
```

```
CONSTRUCT { [[lp:treeNode(?subj)]] ?prop [[lp:treeNode(?obj)]] }
FROM NAMED <liver> [
   CONSTRUCT {dl:Liver dl:part ?o }
   FROM <http://localhost/fma_db>
   WHERE { dl:Liver dl:part ?o }

   UNION

   CONSTRUCT { ?c dl:part ?d }                              (b)
   FROM NAMED <liver>
   FROM NAMED <http://localhost/fma_db>
   WHERE { GRAPH <liver> { ?a ?b ?c } .
             GRAPH <http://localhost/fma_db> { ?c dl:part ?d } .
   }
] # liver
WHERE { GRAPH <liver> { ?subj ?prop ?obj } }
```

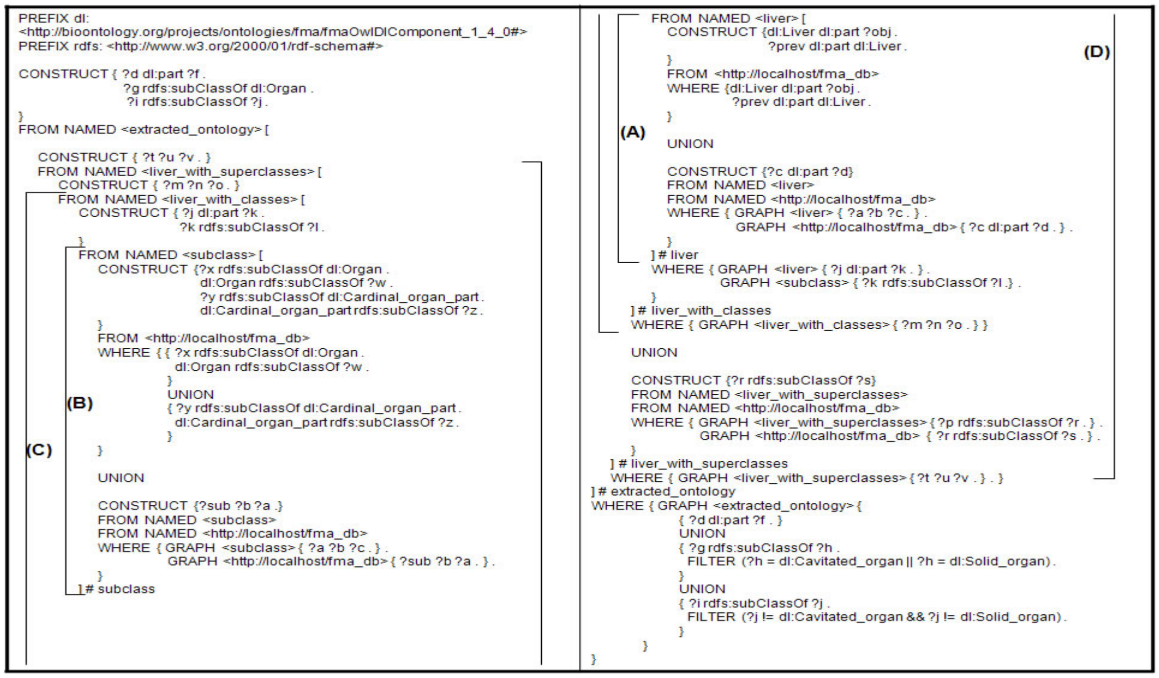
Figure 2: **vSparQL queries.** (a) Subqueries (b) Recursive subqueries and Skolem functions.

query to an existing query, simply associate the existing query with a new datasource. Figure 2(a) presents a query that calculates the same result as Figure 1(c) using subqueries.

Recursive queries in vSparQL are built on top of subqueries. Within a recursive subquery, one or more CONSTRUCT queries are listed; the RDF graphs resulting from each CONSTRUCT query are combined using a set UNION. The data source name associated with a recursive subquery (*liver* in Figure 2(b)) can be used in its constituent CONSTRUCT queries. The result of a recursive subquery is calculated by repeatedly evaluating its subqueries until a fixed point is reached.

Figure 2(b) contains an example of a recursive query that calculates all of the parts of the liver. Within the subquery, the first CONSTRUCT query calculates the base case – all of the direct parts of the liver. The first time that the second CONSTRUCT clause is evaluated, the *liver* data source is empty, and therefore this CONSTRUCT's result graph is empty. On the next iteration, the *liver* data source contains all of the direct parts of the liver; this CONSTRUCT query now returns a graph with all parts of the liver once removed. Each subsequent iteration adds the parts of the liver one step further removed.

vSparQL provides support for the creation of new nodes through the use of Skolem functions. This is particularly useful when combining resources from two ontologies; new resource nodes can be created from the results of queries on the on-



```
PREFIX dl:
<http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT { ?d dl:part ?f .
            ?g rdfs:subClassOf dl:Organ .
            ?i rdfs:subClassOf ?j .
}
FROM NAMED <extracted_ontology> [

   CONSTRUCT { ?t ?u ?v . }
   FROM NAMED <liver_with_superclasses> [
      CONSTRUCT { ?m ?n ?o . }
      FROM NAMED <liver_with_classes> [
         CONSTRUCT { ?j dl:part ?k .
                     ?k rdfs:subClassOf ?l .
         }
         FROM NAMED <subclass> [
            CONSTRUCT {?x rdfs:subClassOf dl:Organ .
                       dl:Organ rdfs:subClassOf ?w .
                       ?y rdfs:subClassOf dl:Cardinal_organ_part .
                       dl:Cardinal_organ_part rdfs:subClassOf ?z .
            }
            FROM <http://localhost/fma_db>
            WHERE { { ?x rdfs:subClassOf dl:Organ .
                      dl:Organ rdfs:subClassOf ?w .
                    }
                    UNION
                    { ?y rdfs:subClassOf dl:Cardinal_organ_part .
                      dl:Cardinal_organ_part rdfs:subClassOf ?z .
                    }
            }

            UNION

            CONSTRUCT {?sub ?b ?a .}
            FROM NAMED <subclass>
            FROM NAMED <http://localhost/fma_db>
            WHERE { GRAPH <subclass> { ?a ?b ?c . } .
                    GRAPH <http://localhost/fma_db> { ?sub ?b ?a . } .
            }
         ]# subclass
         FROM NAMED <liver> [
            CONSTRUCT {dl:Liver dl:part ?obj .
                       ?prev dl:part dl:Liver .
            }
            FROM <http://localhost/fma_db>
            WHERE {dl:Liver dl:part ?obj .
                   ?prev dl:part dl:Liver .
            }

            UNION

            CONSTRUCT {?c dl:part ?d}
            FROM NAMED <liver>
            FROM NAMED <http://localhost/fma_db>
            WHERE { GRAPH <liver> { ?a ?b ?c . } .
                    GRAPH <http://localhost/fma_db> { ?c dl:part ?d . } .
            }
         ] # liver
         WHERE { GRAPH <liver> { ?j dl:part ?k . } .
                 GRAPH <subclass> { ?k rdfs:subClassOf ?l .} .
         }
      ] # liver_with_classes
      WHERE { GRAPH <liver_with_classes> { ?m ?n ?o . } }

      UNION

      CONSTRUCT {?r rdfs:subClassOf ?s}
      FROM NAMED <liver_with_superclasses>
      FROM NAMED <http://localhost/fma_db>
      WHERE { GRAPH <liver_with_superclasses> { ?p rdfs:subClassOf ?r . } .
              GRAPH <http://localhost/fma_db> { ?r rdfs:subClassOf ?s . } .
      }
   ] # liver_with_superclasses
   WHERE { GRAPH <liver_with_superclasses> { ?t ?u ?v . } . }
] # extracted_ontology
WHERE { GRAPH <extracted_ontology> {
            { ?d dl:part ?f . }
            UNION
            { ?g rdfs:subClassOf ?h .
              FILTER (?h = dl:Cavitated_organ || ?h = dl:Solid_organ) .
            }
            UNION
            { ?i rdfs:subClassOf ?j .
              FILTER (?j != dl:Cavitated_organ && ?j != dl:Solid_organ) .
            }
        }
}
```

Figure 3: **Radiologist ontology query.** This query creates the ontology derived from the FMA contains parts of the liver large enough to be seen on an image and the FMA's associated "is_a" hierarchy.

tologies. Skolem functions have the property that a unique node is created for every unique combination of arguments; repeatedly invoking a Skolem function with the same arguments will always return the same node. Skolem functions can occur in any query location that expects a resource node – WHERE clauses, FILTER constraints, and CONSTRUCT templates. Arguments to Skolem functions can be variables or expressions. The syntax for Skolem functions is:

```
[[<skolem_function_url>(arg1, ... )]]
```

Skolem function constructors are specified by URLs. Resource nodes created by Skolem functions are represented in RDF graphs by URLs (with arguments WWW-encoded) of the form:

```
<skolem_function_url>?param1=arg1&param2=arg2
```

For example,

```
[[<http://.../liverOnt/treeNode>(‘‘Y’’)]]
```

would result in the URL

```
http://.../liverOnt/treeNode?param1=\%22Y\%22
```

Figure 2(b) presents a query that delivers all of the parts of the liver; for every part, a new node is constructed in the namespace http://.../liverOnt/ by the Skolem function http://.../liverOnt/treeNode().

Results

We have implemented vSparQL by extending ARQ [18]; ARQ is an open source query processor for SparQL. In this section, we use vSparQL to generate the radiologist's ontology previously described.

Using the FMA as a reference ontology, the query in Figure 3 generates an application ontology that contains the visible parts of the liver and the underlying organization of the human body captured by the FMA's "is_a" hierarchy.

Figure 3 combines a set of subqueries to generate this new ontology. Subquery (A) recursively locates all of the parts of the liver. (B) traverses the FMA's "is_a" hierarchy to identify all classes that are subclasses of "Organ" and "Cardinal organ part." Subquery (C) combines the information from (A) and (B) to ensure that only those liver parts that are subclasses of "Organ" and "Cardinal organ part" are contained in the ontology. This eliminates those liver parts that are too small to be seen on images of the liver; without this restriction, parts as small as cells would be included in our result. Subquery (D) ensures that our application ontology is principled by calculating the superclasses of all of the visible parts of the liver. The FMA's "is_a" hierarchy is the structure around which all information is organized. We capture the portion of the "is_a" hierarchy that is referenced by

```
<http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Right_portal_vein>
   <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#part>
     <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Anterior_branch_of_right_portal_vein> ;
   <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#part>
     <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Caudate_lobe_branch_of_right_portal_vein> ;
   <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#part>
     <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Trunk_of_right_portal_vein> ;
   <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#part>
     <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Posterior_branch_of_right_portal_vein> .

<http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Segment_of_liver>
   <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#part>
     <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Subsegment_of_liver> .

<http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Parenchymatous_organ>
   <http://www.w3.org/2000/01/rdf-schema#subClassOf>
     <http://bioontology.org/projects/ontologies/fma/fmaOwlDlComponent_1_4_0#Organ> .
```

Figure 4: **Radiologist ontology.** A portion of the radiologist ontology returned by our query.

the parts of the liver to ensure that our application ontology has a single root. The outermost subquery replaces the "Cavitated organ" and "Solid organ" classes with their superclass "Organ."

A small portion of our resulting application ontology is shown in Figure 4.

Although our implementation of vSparQL is sufficient for generating application ontologies, evaluating recursive queries over large data sets can take a substantial amount of time. Improving the run time efficiency of our query processor, particularly with respect to recursive queries, is future work. Additionally, we plan to incorporate the path expressions supported by GLEEN [15].

Conclusions

An organizing framework is needed to realize the potential of linking together large numbers of emerging biomedical ontologies. Reference ontologies are a mechanism that could be used to provide the necessary structure for linking together these domain-specific ontologies. In this paper, we have presented vSparQL, an extension to SparQL that will enable application ontologies to be derived from these large, unwieldy sources. We have added our extensions to ARQ and demonstrated their usefulness by generating an application ontology from the FMA.

This work was funded by NIH HL087706.

References

[1] Brinkley J, Suciu D, Detwiler LT, Gennari J, Rosse C. A framework for using reference ontologies as a foundation for the semantic web. In: Proc. of the American Medical Informatics Association. Bethesda, MD; 2006. p. 96–100.

[2] http://www.w3.org/rdf/;.

[3] http://www.w3.org/tr/owl-features/;.

[4] http://www.w3c.org/;.

[5] http://fma.biostr.washington.edu;.

[6] http://protege.stanford.edu/;.

[7] Karvounarakis G, Alexaki S, Christophides V, Plexousakis D, Scholl M. Rql: a declarative query language for rdf. In: Proc. of the Intl. Conference on World Wide Web. New York, NY; 2002. p. 592–603.

[8] http://www.w3.org/tr/rdf-sparql-query/;.

[9] Sintek M, Decker S. Triple - a query, inference, and transformation language for the semantic web. In: Proc. of the First Intl. Semantic Web Conference on The Semantic Web. London, UK: Springer-Verlag; 2002. p. 364–378.

[10] Magkanaraki A, Tannen V, Christophides V, Plexousakis D. Viewing the semantic web through rvl lenses. In: Proc. of the Intl. Semantic Web Conference; 2003. p. 96–112.

[11] Miklós Z, Neumann G, Zdun U, Sintek M. Querying semantic web resources using triple views. In: Proc. of the Second Intl. Semantic Web Conference; 2003. p. 517–532.

[12] Volz R, Oberle D, Studer R. Implementing views for light-weight web ontologies. In: Proc. of the Seventh Intl. Database Engineering and Applications Symposium; 2003. p. 160–170.

[13] Kochut K, Janik M. Sparqler: Extended sparql for semantic association discovery. In: Proc. of the Fourth European Semantic Web Conference; 2007. p. 145–159.

[14] Alkhateeb F, Baget JF, Euzenat J. Rdf with regular expressions; 2007. `http://hal.inria.fr/inria-00144922/en`.

[15] Detwiler LT, Suciu D, Brinkley J. Regular paths in sparql: Querying the nci thesaurus. In: Proc. of American Medical Informatics Association. Washington DC; 2008. .

[16] Noy NF, Musen MA. Specifying ontology views by traversal. In: Proc. of the Intl. Semantic Web Conference; 2004. p. 713–725.

[17] Schenk S. A sparql semantics based on datalog. In: KI 2007: Advances in Artificial Intelligence. Springer Berlin / Heidelberg; 2007. p. 160–174.

[18] http://jena.sourceforge.net/arq/;.