

# Query Processing in a Mediator based Framework for Linked Data Integration

Vânia M. P. Vidal<sup>1</sup>, José A. F. de Macêdo<sup>1</sup>, João C. Pinheiro<sup>1</sup>, Marco A. Casanova<sup>2</sup>, Fábio Porto<sup>3</sup>

<sup>1</sup>Department of Computing, Federal University of Ceará – Fortaleza, CE – Brazil

<sup>2</sup>Department of Informatics – PUC-Rio – Rio de Janeiro, RJ – Brazil

<sup>3</sup>National Laboratory of Scientific Computing – LNCC– Petrópolis, RJ – Brazil

{vvidal, jose.macedo, joaoslz}@lia.ufc.br, casanova@inf.puc-rio.br, fporto@lncc.br

**Abstract.** *In this paper, we present a three-level mediator based framework for linked data integration. In our approach, the mediated schema is represented by a domain ontology, which provides a conceptual representation of the application. Each relevant data source is described by a source ontology, published on the Web according to the Linked Data principles, thereby becoming part of the Web of linked data. Each source ontology is rewritten as an application ontology, whose vocabulary is restricted to be a subset of the vocabulary of the domain ontology. The three-level architecture permits dividing the mapping definition in two stages: local mappings and mediated mappings. Due to this architecture the problem of query answering can also be broken into two steps. First, the query is decomposed, using the mediated mappings, into a set of elementary sub-queries expressed in terms of the application ontologies. Then, these sub-queries are rewritten, using the local mappings, in terms of their local sources schemas. The main contribution of the paper is an algorithm for reformulating a user query into sub-queries over the data sources. The reformulation algorithm exploits inter-ontology links to return more complete query results. Our approach is illustrated by an example of a virtual store mediating access to online booksellers.*

## 1. INTRODUCTION

The Semantic Web is attempting to provide technologies for effectively publishing, retrieving and integrating RDF data distributed over the web. We agree with [15] that large-scale data integration is probably one of the best use cases for the Semantic Web technology. There are several aspects of the Semantic Web that make it appropriate for the integration of data from distributed and heterogeneous data sources [26]. Briefly, these are: RDF, the Resource Description Framework, a simple, but powerful and extensible data model; URIs (or IRIs) used for global naming; and the possibility of reasoning based on Description Logic [6].

In this paper, we consider the problem of designing data integration systems [16] when the data sources are published on the Web according to the Linked Data principles [5], which require the identification of entities with URI references that can be resolved over the HTTP protocol into RDF data that describes the identified entity. These descriptions may include RDF links pointing to other data sources. RDF links take the form of RDF triples, where the subject of the triple is an URI reference in the namespace of one data source, while the object is a URI reference in the namespace of the other. The notion of identity is an important issue in the Semantic Web. URIs guarantee that resources are uniquely identifiable resources on the Web, but they do not guarantee the uniqueness of the entities the resources refer to [11]. Thus, there is a need for a service that is able to find different URIs that refer to the same real-world entity.

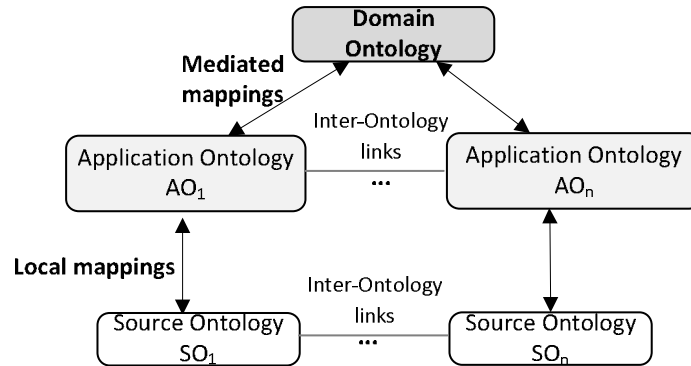
In this paper, we propose a Mediator-based framework for implementing data integration over linked data. We provide a sound and complete algorithm for reformulating a SPARQL query into a query over the (linked) data sources. The reformulation algorithm exploits inter-ontology links to return more complete query results.

This paper is organized as follows. Section 2 describes the framework proposed for linked data integration. Section 3 summarizes some basic concepts required in the paper. Section 4 presents an example that will be used throughout the paper. Section 5 discusses the query answering method adopted. Section 6 introduces a strategy for query reformulation, which is the central contribution of the paper. Section 7 lists related work. Finally, Section 8 presents the conclusions and directions for future research.

## **2. A FRAMEWORK FOR LINKED DATA INTEGRATION**

In this section, we discuss the three-level architecture for linked data integration, which is depicted in Figure 1.

The mediated schema is represented by a domain ontology (DO), which provides a conceptual representation of the domain (a globally shared vocabulary and a set of constraints). Each relevant data source is described by a source ontology, published on the Web according to the Linked Data principles, thereby becoming part of the Web of linked data. These source ontologies are depicted in the Web of Linked Data layer in Figure 1.



**Figure 1. Three-Level Architecture for Linked Data Integration.**

The local source schemas are accessed via wrappers, like those introduced in [3], which export the local data into OWL. Each source ontology is rewritten as an application ontology (AO), whose vocabulary is restricted to be a subset of the vocabulary of the domain ontology. In [24], we present a strategy to automatically generate such application ontologies, considering a set of local ontologies, a domain ontology and the result of the matching between each local ontology and the domain ontology. We adopt OWL Lite [2] as the ontology language to represent the domain ontology, the source ontologies and the application ontologies.

In our framework, the application ontologies help breaking the query answering problem. They are also a notational convenience to divide the definition of the mappings into two stages: the definition of the *mediated mappings* and the definition of the *local mappings*, thereby facilitating the query rewriting process [25]. The mediated mappings specify the concepts of the domain ontology in terms of the application ontologies; whereas the local mappings define the concepts of the application ontologies in terms of the elements of their corresponding local source ontology. Application ontologies enable the identification and the association of semantically corresponding concepts, so they are useful for enhancing tasks such as information discovery and retrieval, and also data integration. We remark that Lutz [17] also adopts this architecture.

### 3. BASIC DEFINITIONS

#### 3.1 RDF and OWL

RDF [19] is a general model language, optimized for data sharing and interchange. The easiness of data interchange arises from some characteristics of this language, like the RDF graph structure, the simple structure of the basic units of these graphs, and the global namespace provided by the use of IRIs (*Internationalized Resource Identifiers*).

In RDF, a data item is represented as *RDF statement* (or simply a *statement*), which is a triple  $(s, p, o)$ , where  $s$  is a IRI, called the *subject* of the statement,  $p$  is a IRI, called the *property* of the statement, and  $o$  is either a IRI or a literal, called the *object* of the statement; if  $o$  is a literal, then  $o$  is also called the *value* of property  $p$ . Each IRI and literal has a global scope. The use of global names is critically important, because it means that the triples can always be merged without name translations. Since each part of the statement in a graph can be used without translation, entire graphs can be

transported and combined without any translation, which is a great advantage when exchanging data.

The *Web Ontology Language* (OWL) [2] describes classes and properties in a way that facilitates machine interpretation of Web content. An *OWL schema* is an ontological description that may be serialized into a collection of RDF triples. A concept of an OWL schema is a class, datatype property or object property. The vocabulary of the schema is the set of concepts defined in the schema (a set of IRIs). The scope of a property name is global to the OWL schema, and not local to the class indicated as its domain. The OWL language family is organized as three dialects: OWL Lite, OWL DL and OWL Full.

In this work, we use OWL Lite. Briefly, this dialect supports named classes, datatype and object properties, subclasses, and individuals. The domain of a datatype or object property is a class, the range of a datatype property is an XML schema type, whereas the range of an object property is a class. As property restrictions, the dialect admits *minCardinality* and *maxCardinality*, with the usual meaning; and *InverseFunctionalProperty*, which resemble the notion of a simple key in databases, for object properties. The *InverseFunctionalProperty* assigned to a property  $p$ , with domain  $C$  and range  $D$ , specifies that given two instances  $C(x1)$  and  $C(x2)$ , such that  $p(x1, y1)$  and  $p(x2, y2)$ , with  $D(y1)$  and  $D(y2)$ , if  $y1=y2$  then  $x1=x2$ .

Finally, we say that: a property  $p$  is an *inter-ontology property* or an *inter-ontology link* iff the domain and range of  $p$  belong to different ontology;  $p$  is an *inter-ontology link* between  $V$  and  $U$  iff  $V$  is the vocabulary of the domain and  $U$  is the vocabulary of the range of  $p$ ;  $p$  is an *inter-ontology link to  $W$*  iff  $W$  is the vocabulary of the domain or of the range of  $p$ ;  $p$  is an *inter-ontology property* between  $V$  and  $U$  iff  $p$  is a *InverseFunctionalProperty* in  $V$  and in  $U$ .

We define two sets, namely *uri-link* and *same-as*, containing information about inter-ontology property and inter-ontology link, respectively. *URI\_link* is a set of triples  $(oi, oj, p)$ , where each  $oi$  and  $oj$  are distinct application ontology and  $p$  is a ontology property such that  $dom(p) \in oi$  and  $range(p) \in oj$ . Similarly, *same-as* is a set of triples  $(oi, oj, p)$ , where  $oi$  and  $oj$  are distinct application ontology and  $p$  is a common property in both ontologies  $oi$  and  $oj$ , such that  $p \in oi, oj$ .

### 3.2. SPARQL Query Language

SPARQL (*SPARQL Protocol and RDF Query Language*) [21] is a W3C standard recommendation. It is a declarative query language that allows extracting data from RDF graphs based on a graph pattern matching, whose basic constructs are *triple patterns*. An example of a triple pattern is  $(?book \textit{s:title}, ?t)$ .

As an example, consider the following SPARQL query (applied over the Publishers application ontology presented in Figure 6):

```

PREFIX pub: <http://publishers/>
SELECT ?n, ?ph
FROM <publishers.owl>
WHERE {
  ?p s:name ?n .
  OPTIONAL {?p s:phone ?ph } .
  ?p s:country ?c .
  FILTER regex(?c, "USA") .
} ORDER BY ?n ?ph

```

**Figure 2. Simple SPARQL query.**

In Figure 2, *?name* and *?phone* are variables, and the prefix ‘*pub*’ identifies the dataset against which the query will be executed. The operator *AND* (denoted as “.”) is equivalent to the relational *JOIN* operator, while the operator *OPTIONAL* is very similar to the relational *LEFT-OUTER-JOIN* operator [20]. More precisely, the *OPTIONAL* operator joins its inner expression with the outer one; thereby holding outer result mappings for which no join partner exists. Note that the keyword *FILTER* is a restriction for a specified condition, and that the keyword *regex* defines an operation to test strings that is based on regular expressions. Finally, the keyword *ORDER BY* specifies a sorted result list. Thus, this query retrieves the names and phones of all publishers, whose country is *USA*.

### 3.3. Rule-based Mapping Formalism

Let  $\mathcal{F}$  be a set of function symbols and  $\mathcal{V}$  be a set of variables. A *constant* is a 0-ary function symbol. The set of *terms* over  $\mathcal{F}$  and  $\mathcal{V}$  is recursively defined as follows: (i) each variable  $v$  in  $\mathcal{V}$  is a term; (ii) each constant  $c$  in  $\mathcal{F}$  is a term; (iii) if  $t_1, \dots, t_n$  are terms, and  $f$  is an  $n$ -ary function symbol in  $\mathcal{F}$ , then  $f(t_1, \dots, t_n)$  is a term. An *atom* over  $\mathcal{F}$ ,  $\mathcal{P}$  and  $\mathcal{V}$  is an expression of the form  $c(t)$ , where  $c$  is a atomic concept and  $t$  is a term, or of the form  $p(t, u)$ , where  $p$  is an atomic role and  $t$  and  $u$  are terms.

Let  $O_S$  and  $O_T$  be two ontologies. A *class mapping* is specified through a set of *mapping rules*, each one of the form

$$\beta_1(w_1) \Leftarrow \alpha_1(v_1), \dots, \alpha_m(v_m)$$

where,  $\alpha_i(v_i)$  is an atom whose atomic concept or atomic role occurs in the source ontology  $O_S$ , for  $i=1, \dots, m$ , and  $\beta_1(w_1)$  is an atom whose atomic concept or atomic role occurs in the target ontology  $O_T$ . We say that  $\alpha_1(v_1), \dots, \alpha_m(v_m)$  is the *body* of the mapping rule, understood as a conjunction, and  $\beta_1(w_1)$  is the *head* of the mapping.

A *property mapping* is likewise specified, except that the head of the rule is a term of the form  $p(t, u)$ , where  $p$  is an atomic role and  $t$  and  $u$  are terms.

A *virtual class* (or *virtual property*) is a class (or property) that occurs in the head or a rule.

This rule-based formalism supports *Skolem functions* [13] for the creation of *new object identifiers* of classes in  $O_T$  from one or more properties of  $O_S$ . In this paper, the Skolem functions are simply used as URIref generators. So, these mapping rules allow the construction of URIrefs for new objects in  $O_T$  as terms of the form  $f(t_1, \dots, t_n)$ , where  $f$  is an  $n$ -ary function symbol and  $t_1, \dots, t_n$  is a sequence of terms of  $O_S$ . Indeed, heterogeneous mappings [9], that use Skolem functions, are necessary to

express the semantic relationships between two ontologies, when, for example, the information represented as a class in one ontology is represented as an object property in the other.

Finally, to simplify the notation, when two or more rules have the same head, we combine their bodies into a single expression with the help of semi-colons. For example,

$$s:title(t) \Leftarrow ap:title(t); ep:title(t)$$

denotes the two rules

$$\begin{aligned} s:title(t) &\Leftarrow ap:title(t) \\ s:title(t) &\Leftarrow ep:title(t) \end{aligned}$$

#### 4. RUNNING EXAMPLE

This section presents an example, adapted from [7], of a virtual store mediating access to online booksellers.

**Domain Ontology:** We assume that the user provides a domain ontology. Figure 3 shows a conceptual representation of the Sales domain ontology. We use the namespace prefix “s:” to refer to the vocabulary of this domain ontology. For example, *s:title* is defined as a datatype property with domain *s:Product* and range *string*, *s:Book* is declared as a subclass of *s:Product*, and *s:hasPub* is defined as an object property with domain *s:Book* and range *s:Publ*. Figure 4 shows the *inverse functional axioms* of the Sales domain ontology.

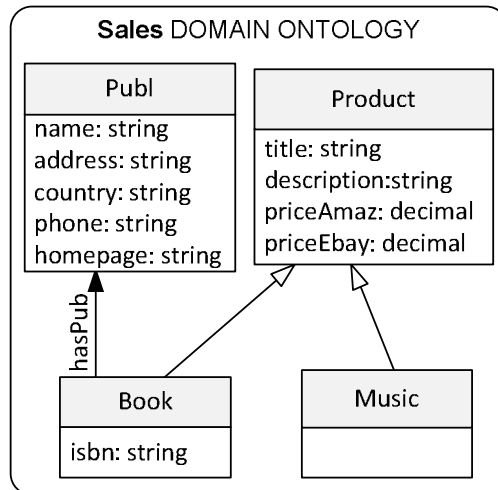


Figure 3. Domain Ontology.

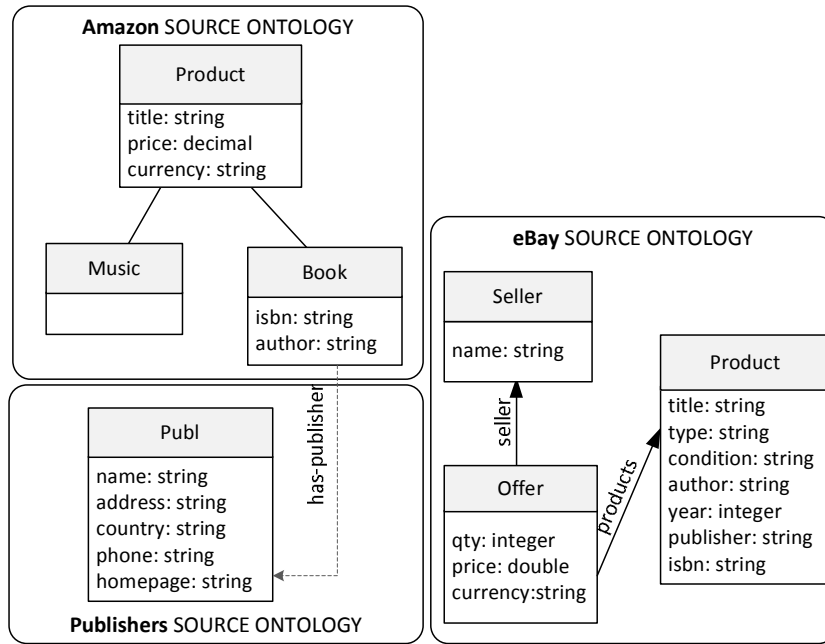
<pre>A1: (s:name rdf:type owl:InverseFunctionalProperty) A2: (s:title rdf:type owl:InverseFunctionalProperty)</pre>
---

Figure 4. Inverse Functional Axioms.

**Source Ontologies:** We assume that we have two source ontologies describing data about Amazon and eBay virtual stores, and a third application ontology describing book Publishers. Figure 5 shows a conceptual representation of the source ontologies. We use

the namespace prefixes “*am:*”, “*eb:*” and “*pub:*” to refer to the vocabularies of *Amazon*, *eBay* and *Publishers* source ontologies, respectively.

Notice that the property *eb:publisher* with domain *eb:Product* and range *pub:Publ* is an inter-ontology property because it connects resources from different ontologies. The instances of the property *eb:publisher* are RDF links that connects an instance of *eb:product* with an instance of *p:publ*.



**Figure 5. Source Ontologies.**

**Application ontologies:** Figure 6 shows the applications ontologies which are obtained based on the result of the matching between each local ontology and the domain ontology. The vocabulary of an application ontology consists of the classes and properties which are, intuitively, the subset of the domain ontology that matches the source ontology. We use the namespace prefixes “*ap:*”, “*ep:*” and “*pp:*” to refer to the vocabularies of *Amazon*, *eBay* and *Publishers* application ontologies, respectively.

The inverse functional axiom *A1* specifies a virtual *same-as* property relating instances of *ap:Publ* and *pp:Publ*, defined as follows:

$$owl:same-as(X, Y) \leftarrow ap:Publ(X), ap:name(X, n), pp:Publ(Y), pp:name(Y, n)$$

Likewise, the inverse functional axiom *A2* specifies a virtual *same-as* property relating instances of *ap:Product* and *eb:Product*, defined as follows:

$$owl:same-as(X, Y) \leftarrow ap:Product(X), ap:title(X, t), ep:Product(Y), ep:title(Y, t)$$

**Local Mappings:** Figure 7(a) and 7(b) show the local mappings.

**Mediated Mappings.** Figure 8 shows the mediated mappings (note the use of “;” to denote disjunctive rules).

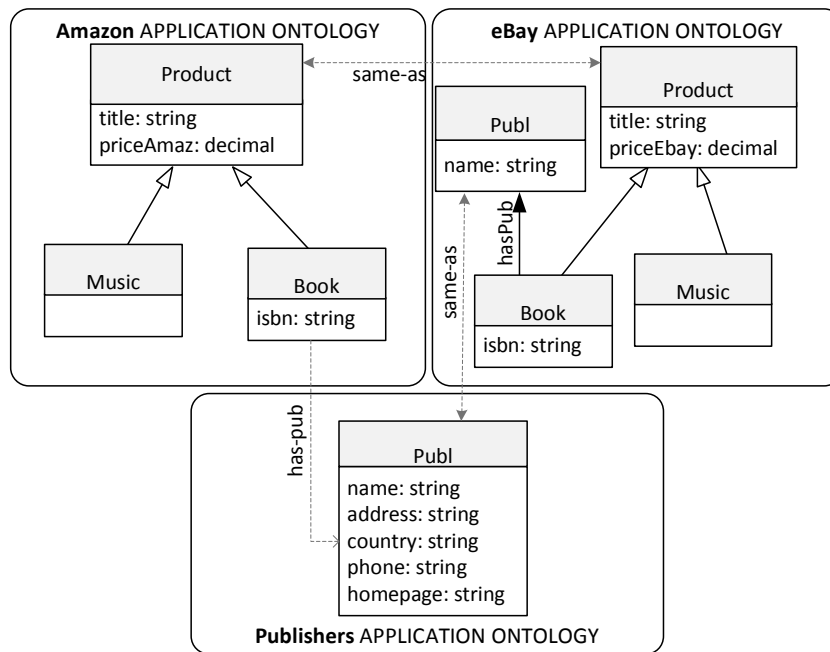


Figure 6. Application ontologies.

```

#1: s:Book(b) ← am:Book(b)
#2: s:Product(b) ← am:Book(b)
#3: s:Music(m) ← am:Music(m)
#4: s:Product(m) ← am:Music(m)
#6: s:title(b,t) ← am:title(b,t), am:Book(b)
#7: s:hasPub(b,p) ← am:has-publisher(b,p), am:Book(b)
#8: s:title(m,t) ← am:title(m,t), am:Music(m)
#9: s:name(p,n) ← am:name(p,n), am:Publ(p)
#10: s:address(p,a) ← am:address(p,a), am:Publ(p)

```

Figure 7(a). Mapping rules from the *Amazon* source ontology to the *Sales* domain ontology.

```

#1: s:Book(p) ← eb:Product(p), eb:type(p)= 'book'
#2: s:Product(p) ← eb:Product(p), eb:type(p)= 'book'
#3: s:Music(p) ← eb:Product(p), eb:type(p)= 'music'
#4: s:Product(p) ← eb:Product(p), eb:type(p)= 'music'
#5: s:title(p,t) ← eb:title(p,t), eb:Product(p), eb:type(p)= 'book'
#6: s:title(p,t) ← eb:title(p,t), eb:Product(p), eb:type(p)= 'music'
#7: s:Publ(fpubl(n)) ← eb:publisher(b,n), eb:Product(b), eb:type(b)= 'book'
#8: s:name(fpubl(n),n) ← eb:publisher(b,n), eb:Product(b), eb:type(b)= 'book'
#9: s:hasPub(b,fpubl(n)) ← eb:publisher(b,n), eb:Product(b), eb:type(b)= 'book'

```

Figure 7(b). Mapping rules from the *eBay* source ontology to the *Sales* domain ontology.

<p><b>Class Mappings:</b></p> <p>s:Product(p) ← ap:Product(p); ep:Product(p)</p> <p>s:Book(b) ← ap:Book(b); ep:Book(b)</p> <p>...</p>	<p><b>Property Mappings:</b></p> <p>s:title(t) ← ap:title(t); ep:title(t)</p> <p>s:name(n) ← ap:name(n); ep:name(n)</p>
---	---

Figure 8. Some of the mediated mappings.

## 5. QUERY ANSWERING METHOD

The ultimate goal of a data integration system is to answer queries posed by the user in terms of the mediated schema (or Domain Ontology). We adopt the SPARQL query language [21] for posing queries on the domain ontology.

The proposed query processing strategy consists of four steps, summarized as follows:

1. **Translation.** The user starts by posing a SPARQL query  $Q$ , expressed in terms of a domain ontology, which is transformed into a parse tree representing the structure of the query in a useful way.
2. **Reformulation.** The query reformulation step is broken into two sub-steps:
  - a. Query  $Q$  is reformulated, based on the mediated mappings, into a set of sub-queries over the application ontologies. Each sub-query aims at extracting data from a single Application Ontology. All concepts that are relevant for answering the query is discovered in this step.
  - b. Each sub-query is then reformulated, based on the local mappings, in terms of a query over the source ontology.
3. **Optimization.** This step attempts to find the most desirable execution plan, according to the following objectives:
  - a. The potential cost reduction obtained by parallelizing sub-query processing. We provide a parallel query execution strategy based on *symmetric hash joins* [1] to deal with the fluctuations on data sources access in the Linked Data environment.
  - b. The potential cost reduction obtained by moving data from one node to another, using a strategy similar to relational *semi-joins* [14]. We provide a *set bind join* operator based on the *bind-join* [10] operator.
4. **Evaluation.** Each sub-query over a source ontology is evaluated at the local data source. The data source is accessed via a Linked Data wrapper, such as the D2R Server [4]. The results of the sub-query are returned to the mediator, where the final result is built according to the optimized execution plan.

We now give an example of how a SPARQL query is processed.

**Example 5.1:** Consider a query  $Q$  over the *Sales* domain ontology that asks for titles and names of their publishers for all books whose publishers' country is "USA". Figure 9 shows this query in SPARQL syntax:

```

PREFIX s:<http:// sales/>

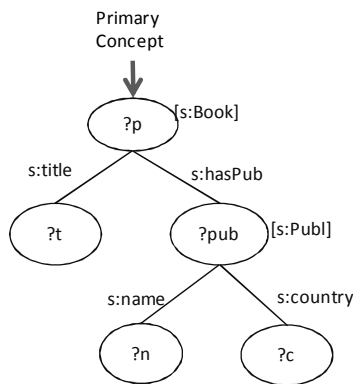
PREFIX rdf:<...>
SELECT ?t
WHERE
{ ?p rdf:type s:Book.
  ?p s:title ?t .
  ?p s:hasPub ?pub .
  ?pub s:country ?c
  FILTER regex(?c, "USA") .
}

```

**Figure 9.** Query  $Q$  in SPARQL syntax.

Each processing step of the SPARQL query  $Q$  is illustrated as follows:

1. **Translation.** The query is parsed, that is, transformed into a query tree representing the structure of the query, as illustrated in Figure 10. Each node of this tree is labeled with a datatype or object variable. Each edge is labeled with a property. Each object variable node has an annotation with respective type. The root node of the tree is called the *primary concept*.



**Figure 10.** Parse tree for query  $Q$ .

2. **Reformulation.**
  - a. First, the query  $Q$  is reformulated in terms of queries over the application ontologies (Figure 11).

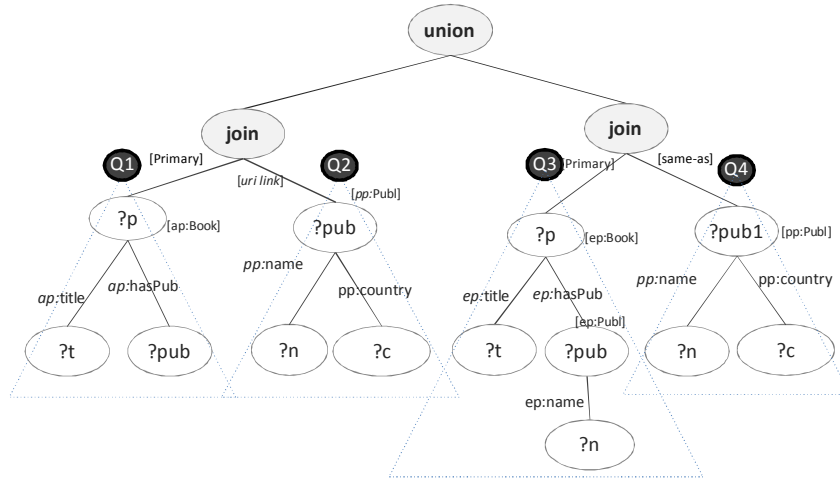


Figure 11. Query Plan.

- b. The query tree in Figure 11 consists of four sub-queries  $Q1$ ,  $Q2$ ,  $Q3$  and  $Q4$ , which aim at extracting data from single application ontology. The generated query tree represents the SPARQL query illustrated in Figure 12.

$Q1$	<pre> <b>SELECT</b> ?t, ?pub <b>FROM</b> http://amazon <b>WHERE</b> {   ?p rdf:type ap:Book .   ?p ap:title ?t .   ?p ap:hasPub ?pub } </pre>	$Q2$	<pre> <b>SELECT</b> ?n, ?pub <b>FROM</b> http://pub1 <b>WHERE</b> {   ?pup pp:name ?n .   ?pup pp:country ?c   <b>FILTER</b> regex(?c,"USA").} </pre>
$Q3$	<pre> <b>SELECT</b> ?t, ?n <b>FROM</b> http://eBay <b>WHERE</b> {   ?p rdf:type ep:Book .   ?p ep:title ?t .   ?p ep:hasPub ?pub .   ?pub ep:name ?n .} </pre>	$Q4$	<pre> <b>SELECT</b> ?n <b>FROM</b> http://pub1 <b>WHERE</b> {   ?pup1 pp:name ?n .   ?pup1 pp:country ?c   <b>FILTER</b> regex(?c,"USA").} </pre>

Figure 12. SPARQL queries over application ontologies.

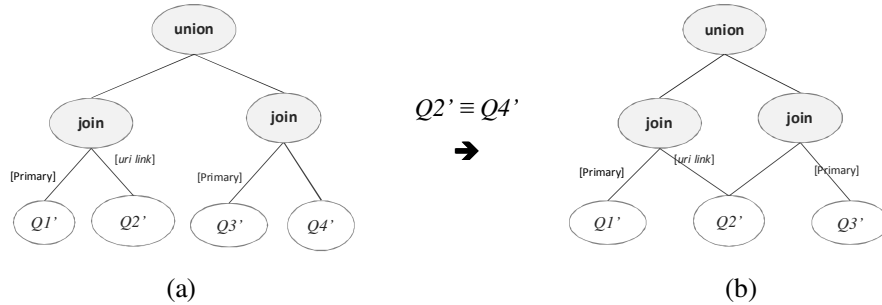
- c. Then the sub-queries are reformulated, based on the local mappings in Figure 7, in terms of a query over the data source schema (Figure 13). The reformulation is unambiguous and therefore straightforward.

$Q1'$	<pre> <b>SELECT</b> ?t, ?pub <b>FROM</b> http://amazon <b>WHERE</b> {   ?p rdf:type am:Book .   ?p am:title ?t .   ?p am:hasPub ?pub } </pre>	$Q2'$	<pre> <b>SELECT</b> ?n, ?pub <b>FROM</b> http://publisher <b>WHERE</b> {   ?pub pub:name ?n .   ?pub pub:country ?c   <b>FILTER</b> regex(?c,"USA").} </pre>
$Q3'$	<pre> <b>SELECT</b> ?t, ?n <b>FROM</b> http://eBay <b>WHERE</b> {   ?prdf:type eb:Product .   ?p eb:title ?t .   ?p eb:publisher ?n .} </pre>	$Q4'$	<pre> <b>SELECT</b> ?n <b>FROM</b> http://publisher <b>WHERE</b> {   ?pub1 pub:name ?n .   ?pub1 pub:country ?c   <b>FILTER</b> regex(?c,"USA").} </pre>

Figure 13. SPARQL queries over source ontologies.

**3. Optimization.** Figure 14(a) shows a graphical and intuitive representation of the execution plan for the queries shown in Figure 13.

First, equivalent queries are identified to avoid fetching the same data more than once. Note that queries  $Q2'$  and  $Q4'$  are similar. Hence, the execution plan is rearranged, as shown in Figure 14(b).



**Figure 14: (a) query plan tree; (b) equivalent plan tree without duplicate queries.**

In this example, the join operator is implemented as a *set bind join*. The latter adapts the *bind join* operator to the data integration context. Considering that each data source is capable of answering SPARQL queries, the *set bind join* corresponds to rewriting the data source subquery adding bind values to join predicates and submitting to the data source.

Our algorithm essentially consists of the following three phases:

(1) **Scan phase**, which performs bind pattern projections on join properties, with duplicate elimination. In our example,  $Q2'$  (Figure 15) projects the name ( $?n$ ) and publisher URI-link ( $?pub$ ) of publishers located in 'USA'.

```

SELECT distinct ?n, ?pub
FROM http://publisher
WHERE {
  ?pub pub:name ?n .
  ?pub pub:country ?c
  FILTER regex(?c, "USA").
}

```

**Figure 15. SPARQL queries over Publisher Source ontologies.**

(2) **Set bind processing phase.**

2.1 *The bind/invoke phase*, which executes several bind joins in parallel to decrease the size of intermediate results and speed up query evaluation. A constraint (SPARQL FILTER expression) is used to specify which data must be joined. Note that sub-queries  $Q1'$  and  $Q2'$  are rewritten adding the bind patterns as filter into sub-queries  $Q1''$  and  $Q2''$  (see Figure 16).

$Q_1$ " (Amazon)	$Q_3$ " (eBay)
<pre> SELECT ?t, ?pub  FROM http://amazon WHERE {   ?p rdf:type am:Book .   ?p am:title ?t .   ?p am:hasPub ?pub   FILTER (     (?pub='http://publisher/p1#')        (?pub='http://publisher/p2#')) } </pre>	<pre> SELECT ?t, ?n FROM http://eBay WHERE {   ?p rdf:type eb:Product .   ?p eb:title ?t .   ?p eb:publisher ?n .   FILTER (     (?n = 'Addison-Wesley')        (?n = 'Mit-press')) } </pre>

Figure 16. Variable bindings in SPARQL with FILTER expression.

2.2 *The transmission phase*, which sends to the mediator data previously obtained, in a parallel asynchronous fashion. In our example, the results returned to mediator from queries  $Q_1$ " and  $Q_3$ " are: "Fundamentals of Database Systems" and "A Semantic Web Primer".

(3) *Final Processing*, which transmits to the mediator all data needed to compute the query answer. In our example, the final result is the union of the results of  $Q_1$ " and  $Q_3$ ", computed in pipeline by reading the HTTP response stream. The final result would be: "Fundamentals of Database Systems, A Semantic Web Primer".

4. *Evaluation*, which transforms the plan that the optimizer produces by into a *query executable plan*. Currently, we support two join implementations: *symmetric hash join* and *set bind join*.

## 6. QUERY REFORMULATION

In this section, we first discuss the idea of using inter-ontology property links during query answering to overcome incompleteness. Then, we present our query reformulation algorithm that takes into account inter-ontology links for rewriting queries over the application ontologies. The subsequent step to the query reformulation algorithm is the mapping of rewritten queries over ontology sources. This local mapping can be applied through a simple unfolding mechanism. Despite its importance, the algorithm to accomplish this simple unfolding is not part of the scope of this article.

### 6.1 The role of inter-ontology links in query answering

An important issue related to linked data integration is that, in the presence of *same-as* inter-ontology properties, the mediated mapping does not completely specifies the domain ontology data. Therefore, a simple unfolding strategy is in general not sufficient for providing all correct answers, as illustrated in the following example.

**Example 6.1:** Referring to our case study, consider the source ontology instances in Figure 17. Figure 18 shows the application ontologies instances obtained by applying the mappings in Figure 7. Figure 19 shows the instance of the domain ontology Sales obtained by applying the mediated mappings in Figure 8.

Consider the query in Figure 9. Referring to the instance of the domain ontology in Figure 19, the query returns only one title "Fundamentals of Database Systems". But the correct answer for the query is "Fundamentals of Database Systems, A Semantic

*Web Primer*” because, based on the virtual same-as property, we can infer that that  $P1$  and  $P2$  are the same entity (recall that we say that the same-as property is virtual because it is derived by applying a rule, as discussed in Section 3).

Figure 12 shows the query obtained from unfolding the query  $Q$  over the application ontologies on the basis of the mediated mappings in Figure 8. Referring to the data graph in Figure 18, the query returns only one title “*Fundamentals of Database Systems*”. Note that the query in Figure 19 provides all correct answers “*Fundamentals of Database Systems*, *A Semantic Web Primer*”.

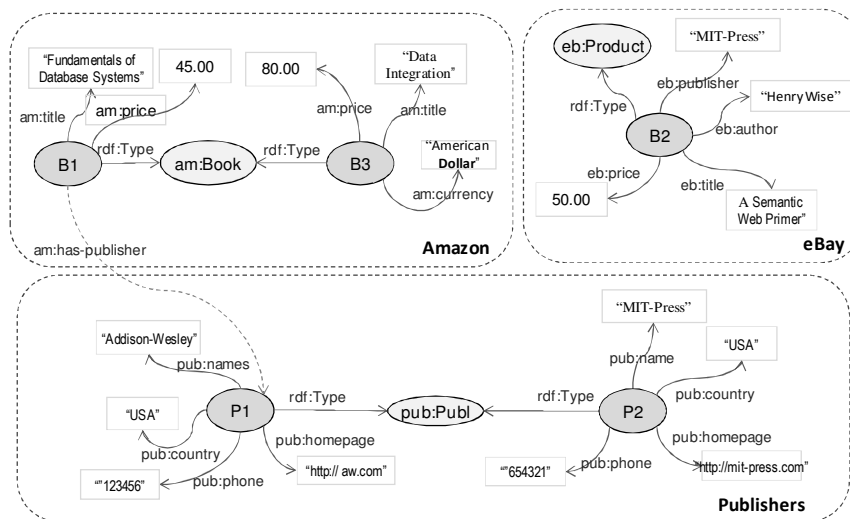


Figure 17. Example RDF data located at three different linked data sources , namely Amazon, eBay and Publishers.

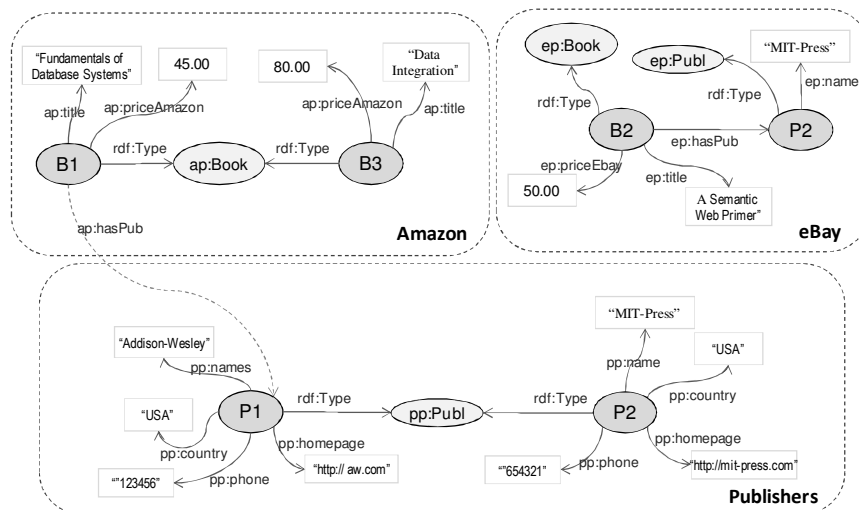


Figure 18. Example RDF data for Application Ontologies: Amazon, eBay and Publishers.

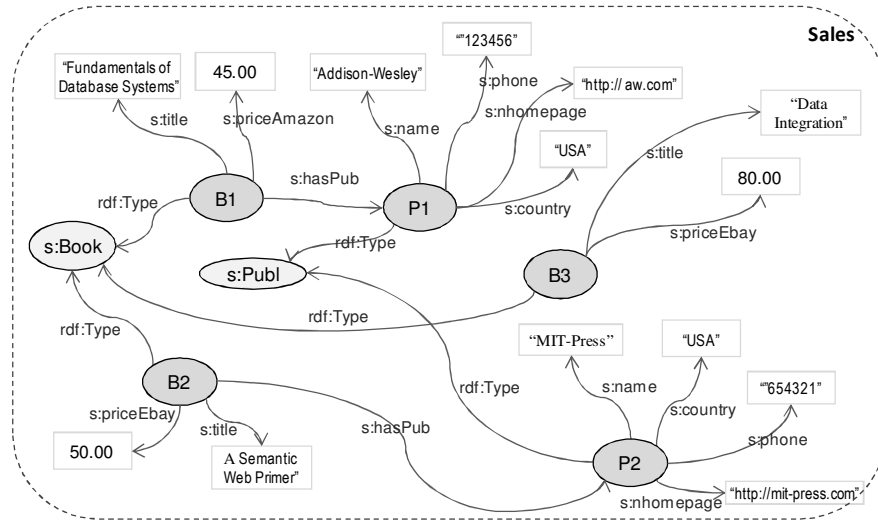


Figure 19. Example RDF data for Sales domain ontology.

## 6.2 The Query Reformulation Algorithm

Figure 20 shows the query reformulation algorithm. Briefly, the main steps of the algorithm are:

1. First, select the application ontologies whose vocabularies contain the *primary concept* of the query tree (QT). We say that these are the *primary ontologies*.
2. Then, for each primary ontology vocabulary  $V$ , generate the *reformulated query tree* (RQT), using the recursive REWRITE\_NODE procedure in Figure 20. This procedure rewrites each property of a node  $N$  of a query tree (QT) in terms of  $V$ , or uses inter-ontology properties to discover other vocabularies for rewriting the properties which are not in  $V$ .
3. The final query plan (FQP) consists of the union of the reformulated query plans of the primary ontologies.

The following example illustrates how the algorithm reformulates a query tree.

**Example 6.2.** Consider the query tree (QT) in Figure 10. The primary concept is  $s:Book$ , which occurs in the vocabularies of the Amazon and the eBay application ontologies. To generate the RQT for a primary ontology, the algorithm calls the REWRITE\_NODE procedure with two parameters: the QT with root node  $r$  and the vocabulary of the primary ontology. The procedure recursively transverse each node  $N$  of QT and rewrites each property  $p$  of  $N$ . The procedure considers three different types of properties:

---

## REWRITE\_NODE (r , V)

Rewrites the properties (edges) of a node  $r$  in a query tree with the vocabulary  $V$ , and uses inter-ontology properties to discover other vocabularies for rewriting the properties that are not in  $V$ . The result of REWRITE\_NODE is a reformulated query tree, whose root is a join node with the following children: the primary query tree (Query tree rewritten with the vocabulary  $V$ ) and all the query trees rewritten with other vocabularies for the properties which are not in  $V$ .

---

**input:** root node  $r$  of a query tree QT  
vocabulary  $V$

**output:** reformulated query tree RQT

```
1  Begin
2   $SS \leftarrow \emptyset; RQT \leftarrow QT;$ 
3
4  for each edge  $e(r, next)$  with label  $pe$  whose range is in  $V$  do
5    replace the label  $pe$  of  $e$  by  $V:pe$ 
6    if  $pe$  is an inter-ontology link whose range is in  $Vt$  such that  $Vt \neq V$ 
7      createUriLinkSQT( $r$ )
9      rewrite_Node ( $r_{new}, Vt$ )
10     add  $r_{new}$  to  $SS$ 
11   Else
12     rewrite_Node ( $r_{next}, O$ )
13   end-if
14 end-for
15 let  $E = \{ label(e) | e \text{ is an edge that leaves node } r \text{ and } label(e) \notin V \}$ 
16 for each Vocabulary  $V_i$  such that there exists a virtual same-as link ( $V:cr, V_i:cr, PJ$ ) such that  $E \cap V_i \neq \emptyset$ 
17   createSame-asLinkSQT( $r$ )
18   rewrite_query( $r_{new}, V_i$ )
19   add  $r_{new}$  to  $SS$ 
20 end-for
21 if  $SS \neq \emptyset$ 
22   add  $r$  to  $SS$ 
23    $RQT \leftarrow SS$ 
24 end-if
25 End
```

---

**Figure 20. THE QUERY REFORMULATION ALGORITHM.**

a) When  $p$  is in the vocabulary  $V$  and  $p$  is not an inter-ontology property (line 5), then replace the namespace of  $p$  by  $V$ . For example, in Figure 21, property " $s:name$ " is replaced by " $am:name$ ".

b) When  $p$  is in the vocabulary  $V$  and  $p$  is an inter-ontology property with range of  $p$  in vocabulary  $Vt$  (line 7-10), then a secondary query tree SQT is created with root node  $r_{new}$ , which is a copy of the node  $r_{next}$ , incident to  $p$ ; the children of  $r_{new}$  are all the descendants of  $r_{next}$ . Next, the query tree SQT is rewritten using  $Vt$  namespace. In

our example, property "*s:hasPub*" is an inter-ontology link (*ap:book*, *pp:publ*, *ap:hasPub*).

Figure 21 summarizes the main steps for rewriting property "*s:hasPub*": (1) create the SQT shown in Figure 22; delete all descents of node *?pub* from primary query tree; (3) rewrite SQT with the publisher's (*pub*) namespace. In Figure 21, the join variable *?pub* of the *ap:hasPub* property is used in the SQT tree to implement the URI join between both queries.

c) If *p* is not in the Vocabulary *V* (lines 16-20), then for each virtual *same-as* inter-ontology link (*V:cr*, *Vi:cr*, *PJ*) in *V*, create a secondary query tree SQT which is a copy of the sub-graph containing node *N* and all the descendents of the properties not in *V*. Next, rewrite SQT with *Vi*.

In our example, the property *country* is not in the Amazon vocabulary. Using the virtual *same-as* link (*am:Publ*, *pub:publ*, *pub:name*) we are able to join instances of *am:publ* with instances of *pub:publ* to get the country that is in publisher's vocabulary.

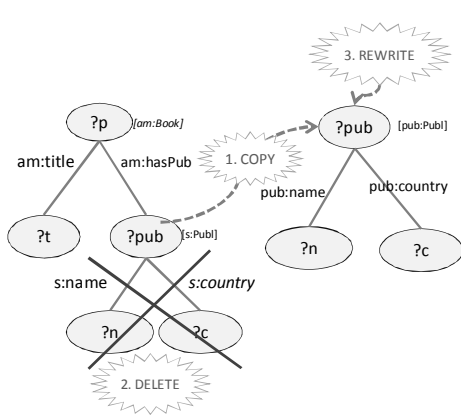
Figure 22 summarizes the main steps for rewriting the property *s:country*: (1) create the SQT show in Figure 22 delete the edge *country* from primary query tree; (3) rewrite the SQT with the Publisher's namespace (*pub*).

Figure 22 shows the secondary query graph for the virtual *same-as* link (*am:Publ*; *pub:publ*, *pub:name*).

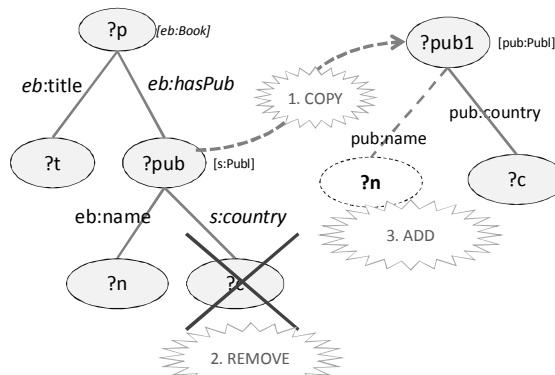
The result of REWRITE\_NODE (*N*, *V*) is a reformulated query tree (RQT) whose root is a join node with the following children: the primary query tree (Query tree rewritten with the vocabulary *V*) and all generated secondary query trees.

Figure 21 shows the reformulated Query Tree for the Amazon vocabulary. The join variable "*?pub*" of *am:hasPub* property is used in the secondary query tree to implement the join operation. Figure 22 shows the Reformulated Query Tree for the eBay vocabulary. The variable "*?name*", corresponding to "*eb:name*" and "*pub:name*" properties, is required (in both query trees) to process the join operation.

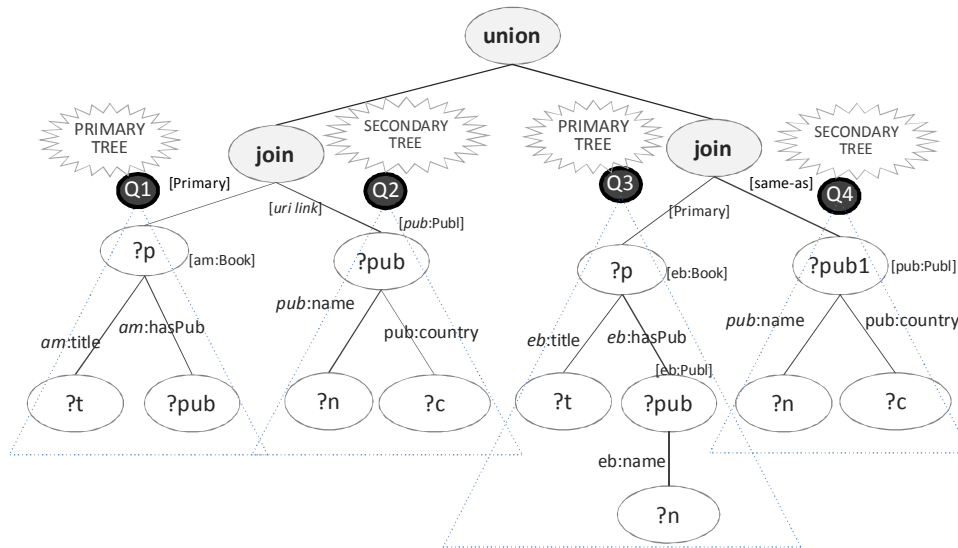
Finally, all join query trees are grouped into a single final query plan (FQP). Figure 23 illustrates the FPQ generated in our example. Note that this query plan generates four queries that should be submitted to three application ontologies.



**Figure 21. Property link processing.**



**Figure 22. Same-as property processing.**



**Figure 23. Final Query Plan.**

## 7. RELATED WORK

Data integration has been a topic of research in the database field for a long time [28].

In [8], it is provided an ontology-based approach to the integration of heterogeneous XML documents that transforms heterogeneous XML sources into local RDF ontologies, which are then merged into a RDF global ontology. Note that this work does not define an approach for the unification of the results that are returned from different data sources.

Some proposals have been provided recently for the integration of data from distributed RDF data sources. In [22], the DARQ engine for federated SPARQL queries is presented. DARQ decomposes a SPARQL query in sub-queries, sends the sub-queries to distributed data sources, and integrates the results of the sub-queries.

The SemWIK system, proposed in [15], is a mediator-based system [27], which provides transparent access to distributed RDF data sources. In SemWIK, the data can be retrieved using SPARQL queries, while RDF schema or OWL ontologies can be used to describe the data sources. Makris *et al.* [18] presents a mediator framework based on OWL and SPARQL. In the proposed framework, query reformulation is based on a set of mapping types.

In the above systems, all relevant data sources that provide queried data are discovered before the query execution. Since inter-ontology links are not considered during query reformulation, the result of the query may be incomplete.

Hartig *et al.* [12] presents an approach to query the Web of Linked Data where the queried data is discovered and retrieved at query execution time. This approach is based on the idea of looking up URIs during query execution time proposed by Berners-Lee *et al.* [3]. According to [12], basically, a SPARQL query is executed by iteratively dereferencing URIs in order to fetch their RDF descriptions from the web, and building solutions from the retrieved data. Although, this centralized approach would return more complete query results, only URI links are considered. Another limitation of that approach is that the query must have a seed URI as the subject of the first query pattern. Besides, optimization techniques for this centralized approach is still a challenge [23].

## 8. CONCLUDING REMARKS AND FUTURE WORK

In this paper, we have presented a three-level mediator based framework for linked data integration over Linked Data environment. This framework takes a query on domain ontology and rewrites it into sub-queries submitted over multiples data sources. The query's result is obtained by the proper combination of data resulting from these sub-queries. We have illustrated, through an example, how our framework allows the combination of data from different linked data sources, thus overcoming some limitations of other ontology-based approaches.

We have focused in this paper on describing a sound and complete algorithm for reformulating a SPARQL query into several queries over the linked data sources (Section 6). An important feature of the reformulation algorithm is the use of inter-ontology links to return more complete query results.

As a future work, we intend to investigate post-processing of instances resulting from query processing and query plan optimization mechanisms. Particularly in instances post-processing, we want to analyse the data fusion problem, which consists in identifying and merging equivalent instances provided by multiples RDF datasets. Concerning query plan optimization, we will define heuristics for generating cost-effective query plan.

## REFERENCES

- [1] Annita N. Wilschut and Peter M. G. Apers. 1993. *Dataflow query execution in a parallel main-memory environment*. Distrib. Parallel Databases 1, 1 (January 1993), 103-128. DOI=10.1007/BF01277522 <http://dx.doi.org/10.1007/BF01277522>
- [2] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. 2004. *OWL web ontology language reference*. W3C Recommendation. Available at: <http://www.w3.org/TR/owl-ref/>.

- [3] Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D. 2006. *Tabulator: Exploring and analyzing linked data on the semantic web*. In: Proceedings of the 3rd Semantic Web User Interaction Workshop (SWUI) at ISWC, November, 2006.
- [4] Bizer, C. and Cyganiak, R. 2006. *D2R Server – Publishing Relational Databases on the Web as SPARQL Endpoints*. In Proceedings of the 15th International World Wide Web Conference.(Edinburgh, Scotland 2006)
- [5] Bizer,C., Heath,T. and Berners-Lee,T. 2009. *Linked Data - The Story So Far*. Int. Journal on Semantic Web and Information Systems, Special Issue on Linked Data, 2009, in press.
- [6] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R. 2008. Data Integration through DL-LiteA Ontologies. 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008): pp. 26-47.
- [7] Casanova, M.A., Lauschner, T., Leme, L.A.P., Breitman, K.K; Furtado, A.L., Vidal, V. M. P. 2009. *A Strategy to Revise the Constraints of the Mediated Schema*. In: Proc. 28th Conf. on Conceptual Modeling, pp. 265-279, Gramado, Brazil.
- [8] Cruz, I. F., Xiao, H., and Hsu, F. 2004. *An Ontology-Based Framework for XML Semantic Integration*. In Proceedings of the international Database Engineering and Applications Symposium (July, 2004). IDEAS. IEEE Computer Society, Washington, DC, 217-226. DOI= <http://dx.doi.org/10.1109/IDEAS.2004.10>.
- [9] Ghidini C., Serafini L. 2006. *Reconciling Concepts and Relations in Heterogeneous Ontologies*. In: Proc. ESWC 2006, pp. 50-64.
- [10]Haas, L. and Kossmann, D. and Wimmers, E. and Yang, J. 1997. *Optimizing Queries across Diverse Data Sources*. In: 23rd International Conference on Very Large Data Bases (VLDB 1997), August 25-29, 1997, Athens, Greece.
- [11]Halpin, .H. 2006. *Identity, Reference, and Meaning on the Web*. In Proceedings of the Workshop on Identity Meaning and the Web (IMW06).
- [12]Hartig, O., Bizer, C., Freytag, J.C. 2009. *Executing SPARQL queries over the web of linked data*. In: Proceedings of the 8th International Semantic Web Conference (ISWC).
- [13]Hull, R., Yoshikawa, M. 1990. *ILOG: Declarative Creation and Manipulation of Object Identifiers*. In: Proc. VLDB 1990, pp. 455-468.
- [14]Kossmann, D. 2000. *The state of the art in distributed query processing*. ACM Comput. Surv. 32, 4 (Dec. 2000), 422-469. DOI= <http://doi.acm.org/10.1145/371578.371598>
- [15]Langegger, A., Wöß, W., Blöchl, M. 2008. *A Semantic Web Middleware for Virtual Data Integration on the Web*. In: Proceedings of the 5th European Semantic Web Conference (ESWC). Volume 5021 of Lecture Notes in Computer Science. Springer Verlag, pp. 493–507.
- [16]Lenzerini, M. 2002. *Data integration: a theoretical perspective*. In Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Madison, Wisconsin, June 03 - 05, 2002). PODS '02. ACM, New York, NY, 233-246. DOI= <http://doi.acm.org/10.1145/543613.543644>.
- [17]Lutz, M. 2006. *Ontology-based Discovery and Composition of Geographic Information Services*. PhD Thesis, Institut für Geoinformatik.
- [18]Makris, K., Bikakis, N., Gioldasis, N., Tsinaraki, C., Christodoulakis, S. 2009. *Towards a mediator based on owl and sparql*. In: WSKS (1). Lecture Notes in Computer Science, vol. 5736, pp. 326–335.
- [19]Manola, F. and Miller, E. 2004. *RDF Primer*. W3C Recommendation, February 2004. Available at: <http://www.w3.org/TR/rdf-primer>.

- [20]Pérez, J., Arenas, M., and Gutierrez, C. 2009. *Semantics and complexity of SPARQL*. ACM Trans. Database Syst. 34, 3 (Aug. 2009), 1-45. DOI=<http://doi.acm.org/10.1145/1567274.1567278>
- [21]Prud'hommeaux, E. and Seaborne, A. 2008. *Sparql Query Language for RDF*. W3C Recommendation. Available at: <http://www.w3.org/TR/rdf-sparql-query/>.
- [22]Quilitz, B. and Leser, U. 2008. *Querying Distributed RDF Data Sources with SPARQL*. In: Proceedings of the 5th European Semantic Web Conference (ESWC). Volume 5021 of Lecture Notes in Computer Science, Springer Verlag, pp. 524–538 (2008). DOI = [http://dx.doi.org/10.1007/978-3-540-68234-9\\_39](http://dx.doi.org/10.1007/978-3-540-68234-9_39).
- [23]Reddy, B. R. K, Kumar, P. S. 2010. *Optimizing SPARQL queries over the Web of Linked Data*. Workshop on Semantic Data Management (SemData@VLDB), September, 2010, Singapore.
- [24]Sacramento, E. R.; Vidal, V. M. P.; Macêdo, J. A.; Lóscio, B. F.; Lopes, F. L. R.; Casanova, M. A.; Lemos, F. 2010. *Towards Automatic Generation of Application ontologies*. To be published in: Proceedings of the 25st Brazilian Symposium on Databases – SBBD, 2010, Belo Horizonte, MG, Brazil.
- [25]Vidal, V.M.P., Sacramento E. R., José A. F., Casanova M. A. 2009. *An Ontology-Based Framework for Geographic Data Integration*. In: Proc. 3rd International Workshop on Semantic and Conceptual Issues in GIS (SeCoGIS 2009), in conjunction with the 28th International Conference on Conceptual Modeling. Berlin / Heidelberg: Springer, 2009.
- [26]Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. and Hübner, S. 2001. *Ontology-based Integration of Information - A Survey of Existing Approaches*. In: Proceedings IJCAI-01 Workshop: Ontologies and Information Sharing, pp. 108-117.
- [27] Wiederhold, Gio. 1992. *Mediators in the Architecture of Future Information Systems*. Computer 25, 3 (March 1992), 38-49. DOI=<http://dx.doi.org/10.1109/2.121508>.
- [28]Ziegler, P., Dittrich, K. R. 2004. *Three Decades of Data Integration - All Problems Solved?*. In Proceedings of 18th IFIP World Computer Congress, Volume 12, Building the Information Society, pp. 3-12