Integrating XML Data Sources using RDF/S Schemas: The ICS-FORTH Semantic Web Integration Middleware (SWIM)

Extended Abstract

Ioanna Koffina[1], Giorgos Serfiotis[1], Vassilis Christophides[1], Val Tannen[2], and Alin Deutsch[3]

[1] Institute of Computer Science, FORTH
Vassilika Vouton P.O 1385 GR 71110, Heraklion, Greece
and
Department of Computer Science, University of Crete
GR 71409, Heraklion, Greece
{koffina,serfioti,christop}@ics.forth.gr

[2] Computer and Information Science Department, UPenn
200 South 33rd Street, Philadelphia, Pennsylvania, USA
val@cis.upenn.edu

[3] Department of Computer Science & Engineering, UCSD
9500 Gilman Drive La Jolla, CA 92093, USA
deutsch@cs.ucsd.edu

1 Introduction

Recent experimental studies highlight the fact that the size of the so-called *Deep Web* [1] (i.e., data stored in legacy databases and accessed through Web forms) largely exceeds the size of the *Surface Web* (i.e., data stored in static HTML pages). Data on the Deep Web are managed under a variety of formats (relational databases or XML) and they are queried using various searching interfaces and languages (i.e., SQL, XPath, XQuery). In this context, a key factor for the success of the Semantic Web (SW) is to provide a useful, comprehensive and high-level access to voluminous data residing in Deep Web sources.

More precisely, SW technology should be able to publish legacy data as valid instances of domain or application specific RDF/S schemas (or other SW ontology language). In other words, we need a SW integration middleware (SWIM) capable either to republish XML as RDF, or to publish RDB data directly as RDF, or even better, capable of performing both.

There are many issues involved in the functionality of a SWIM. In particular, SWIM may act as an integrator for relational and XML sources. Such an integrator should facilitate users to formulate queries against the mediated RDF/S schema using declarative languages (such as RQL [2]), as well as, support further abstraction levels using declarative view definition languages (e.g., RVL [3]). In a nutshell, SWIM should offer the following services: (a) it should be able to establish mapping rules between XML and RDF and between RDB and RDF,

(b) to verify the conformance of these mappings w.r.t the semantics of the employed schemas, (c) to be able to reformulate RDF/S queries against RDB or XML sources, and (d) to combine these queries with RVL views.

In order to address effectively and efficiently the above requirements, we should choose a uniform and expressive logic framework to define SWIM integration services. This framework should exploit background theory on conjunctive queries and query containment and minimization [4].

An architecture based on mediators is highly beneficial for deploying a SWIM. There exist two main approaches for integrating data sources [5] using mediator-based architectures: the Global-as-View (GAV) [6] and the Local-as-View (LAV) [7] [8]. The former provides descriptions of the global schema in terms of the views of local sources and relies on simple query reformulation techniques (i.e., query unfolding). The latter considers local sources as materialized views specified in terms of the global schema. LAV supports easily the evolution of the data integration system by just adding or removing the descriptions of local sources. In our work we advocate a hybrid approach called GLAV [9], which combines the previous advantages and exceeds the expressive power of both GAV and LAV.

2 A Motivating Example

Let's assume an XML source whose content is described by a DTD or an XML Schema (see Figure 1). XML data from this source contains information about Museums, exhibiting some artifacts for which we want to know their creator. Data stored in such sources can be queried using an XML query language like XPath [10] [11] or XQuery [12].

Now suppose that we add on top of this repository an RDF/S schema from the cultural domain. This mediated RDF/S schema can be queried using RQL and it can be used for defining personalized views with the help of RVL. However, since there are no actual RDF data, we need to reformulate the RQL queries expressed against this virtual RDF/S schema into queries appropriate for our XML source. For example, the following RQL query:

```
SELECT X
FROM {X}exhibits{Y}, {Y}denom{Z}
WHERE Z = “Louvre”
```

will be *reformulated* to the following XQuery query:

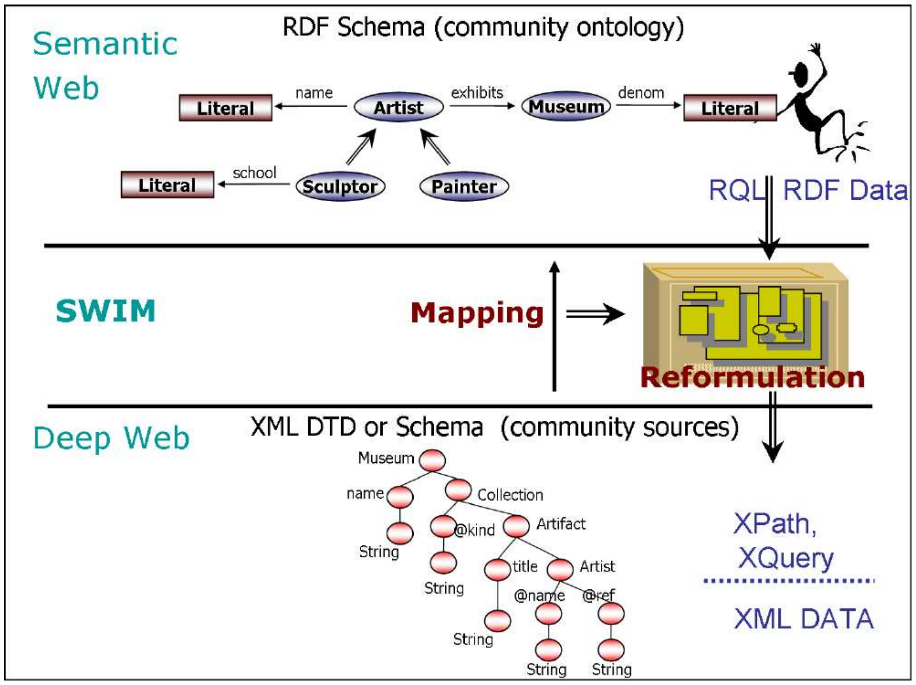

Fig. 1. Republishing XML as RDF

```
<RDF>
{
  <Bag>
  {
    for $var0 in document("art.xml")//Museum
    for $var1 in $var0/name
    for $var2 in $var0/Collection//Artist/@name
    where $var1/text()="Louvre"
    return <li>{$var2/text()}</li>
  }
  </Bag>
}
</RDF>
```

This reformulation involves several challenging issues. First of all, the schemas employed by our XML sources and the RDF/S mediator are different. Their discrepancies, usually called *heterogeneity conflicts*, can be classified under three axes: syntactic, structural and semantic [13]. As we can see in our example (Figure 1), we need to view XML data model through the RDF data model (syntactic conflict), to resolve categorization conflicts, given that in RDF/S there is class hierarchy while in XML there isn't (structural conflict), as well as naming mismatches; for instance the "name" of a Museum in XML is called "denomination" in RDF/S (semantic conflict).

In order to reconciliate the heterogeneous representations of our data we need to define appropriate *mapping rules*. Choosing an expressive and tractable logical framework to map data from XML to RDF/S is crucial. These mappings are used for reformulating queries issued against the virtual RDF/S schema into queries acceptable by our XML sources. However, more complex mappings (in order to increase expressiveness) render query reformulation harder. The reformulation becomes more complex if we take into consideration the presence of constraints capturing the semantics of both RDF/S and XML data model, as well as, application-specific constraints coming from the schema (if any) of XML sources (like keys, foreign keys etc.). Therefore, we need to prove soundness and completeness of our reformulation algorithm.

Since reformulated queries are evaluated to remote sources and mediator queries resulting from automated manipulation/generation may entail redundancies, their optimization is crucial. In particular, optimization tries to simplify queries by removing redundant predicates and to eliminate redundant queries.

3 Contributions

In this context, we propose a middleware called ICS-FORTH SWIM (Semantic Web Integration Middleware) supporting the following functionalities:

3.1 Formal Framework for Mapping Specification

As discussed previously, choosing a logical framework for defining the mappings is of great importance. Our idea was to represent both RDF and XML data models as first order logic predicates and capture their semantics through appropriate constraints. In this way we reduce the RDF to XML query reformulation problem to the relational equivalent one, as well as, reuse existing techniques in relational query containment and minimization.

More precisely, we rely on Linear Datalog (no recursion) for establishing the mappings and translating the RQL/RVL queries and views issued against the virtual RDF/S schema. The head of the Datalog rules consists of a conjunction of view clauses employed by the RDF/S view definition language RVL, and the body consists of XPath atoms that facilitate querying tree-structured XML sources. The former is used in order to point out the instantiation of RDF/S schemas with appropriate resources residing in our XML data sources. Employing some non interpreted built-in predicates (e.g., concat, split) for handling more intricate cases, like complex keys or string manipulation, enhances the expressiveness of the mappings.

As far as these mappings are concerned, they are interpreted in a constraint - oriented way implementing the GLAV approach of our middleware. We can map a *view* over the global RDF/S to a *view* over local XML sources, and each of these mappings is captured with the help of constraints. Constraints describe both the RDF/S schema in terms of the XML sources (GAV) and the XML sources in terms of the global RDF/S schema (LAV).

3.2 Query Reformulation and Optimization

Another powerful functionality of the SWIM is its ability to reformulate queries. RQL queries, expressed in terms of the RDF/S virtual schema, conclude to minimized queries expressed in terms of the XML sources by gradually applying a number of chasing/backchasing [14] [15] [16] steps. Using this algorithm is proven to be sound and complete for disjunction of conjunctive queries in presence of disjunctive embedded dependencies (DEDs), which means that all possible correct results are returned.

Query is chased with the help of the constraints that have been defined to express the semantics of XML and RDF data models and in addition, with constraints (if any) coming from the XML data sources i.e., specification of keys and foreign keys, as well as, of enumerated types. The result of chasing is backchased for producing a minimal reformulation. These minimized queries are simplified (as few predicates as possible) and the redundant ones are eliminated. In this way we guarantee that we query the XML sources with the minimum possible queries. Finally, the minimized reformulated queries are translated into XPath and/or in XQuery.

References

1. The Deep Web: (http://www.brightplanet.com/deepcontent/index.asp)
2. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proceedings of the 11th International World Wide Web Conference (WWW), Honolulu, Hawaii (2002)
3. Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the Semantic Web Through RVL Lenses. In: Proceedings of the Second International Semantic Web Conference (ISWC'03), Sanibel Island, Florida, USA, 20-23 October. (2003)
4. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Publishing Company (1995)
5. Bertossi, L., Bravo, L.: Consistent Query Answers in Virtual Data Integration Systems. In: Inconsistency Tolerance in Knowledgebases, Databases and Software Specifications. Springer (2004)
6. Ullman, J.: Information Integration Using Logical Views. Theoretical Computer Science **239** (2000) 189–210
7. Levy, A.Y.: Logic-Based Techniques in Data Integration. In Jack Minker, ed.: Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14-16, 1999, College Park, Maryland, Computer Science Department, University of Maryland (1999)
8. Levy, A.: Answering Queries Using Views: A Survey. The International Journal on Very Large Data Bases (2001)

9. Friedman, M., Levy, A., Millstein, T.: Navigational Plans for Data Integration. In: Proceedings of the sixteenth national conference on artificial intelligence and eleventh innovation applications of AI conference on Artificial intelligence and innovative applications of artificial intelligence. (1999) 67–73
10. XML Path Language (XPath) 1.0: (http://www.w3.org/tr/xpath/)
11. XML Path Language (XPath) 2.0: (http://www.w3.org/tr/xpath20/)
12. XQuery 1.0: An XML Query Language: (http://www.w3.org/tr/xquery/)
13. Sheth, A., Kashyap, V.: So Far (Schematically) yet So Near (Semantically). In: Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5). (1992)
14. Deutsch, A., Tannen, V.: MARS: A System for Publishing XML from Mixed and Redundant Storage. In: Proceedings of the 29th VLDB Conference, Berlin, Germany. (2003)
15. Deutsch, A., Tannen, V.: Reformulation of XML Queries and Constraints. In: Proceedings of the International Conference on Database Theory (ICDT). (2003)
16. Deutsch, A., Tannen, V.: Querying XML with Mixed and Redundant Storage. Technical report, University of Pennsylvania (2002)