An Ontology-based Framework for XML Semantic Integration

Isabel F. Cruz
ifc@cs.uic.edu
University of Illinois
at Chicago

Huiyong Xiao
hxiao@cs.uic.edu
University of Illinois
at Chicago

Feihong Hsu
fhsu@cs.uic.edu
University of Illinois
at Chicago

Abstract

XML is becoming the standard for data interchange on the web. However, XML and its schema languages do not express semantics but rather structure, such as nesting information. Therefore, semantically equivalent documents often present different document structures. In this paper, we provide an ontology-based framework that aims to make two XML documents interoperate at the semantic level while retaining their nesting structure. In our global-as-view approach, we generate an RDF ontology for each of the participating XML documents, which preserves the nesting structure of the document. An RDF global ontology is the result of merging the individual ontologies. The global ontology unifies the query access and establishes semantic connections among the underlying individual databases. We consider two types of queries: those posed on the global ontology and those that are posed to any of the XML documents, in a P2P fashion. The former type of query is processed using query translation from an RDF query to an XML query. The latter type of query entails bidirectional query processing: the translation from an XML query to an RDF query followed by the translation from an RDF query to an XML query. To ensure the correctness of the answer to the query in the latter case, we introduce the concept of *reversibility* of the query translation.

1 Introduction

1.1 Problem description

One of the primary obstacles in Semantic Web applications is the heterogeneity of the distributed data sources. These heterogeneities can be classified as *syntactic*, *schematic*, and *semantic* heterogeneities [3]. Our previous work gives an example of solving schematic heterogeneities, that is, those that arise from using different schemas to represent the same data [10]. In this paper, we focus on the problem of semantic interoperability between different XML sources. In doing so, we propose an approach for integration of heterogeneous XML sources and query processing across these XML sources.

XML documents that conform to different schemas may represent data with similar semantics. Therefore, a user must construct his queries in accordance to an XML document's structure to retrieve fragments of information that have the same meaning. This fact makes the formulation of queries on heterogeneous XML sources a nontrivial burden to the user. Furthermore, this shortcoming of XML impedes the interoperability of XML sources since the reformulation of XML queries has to eliminate the structural differences of the queries while presenting the same semantics. Let us illustrate the problem using a running example as shown in Figure 1.

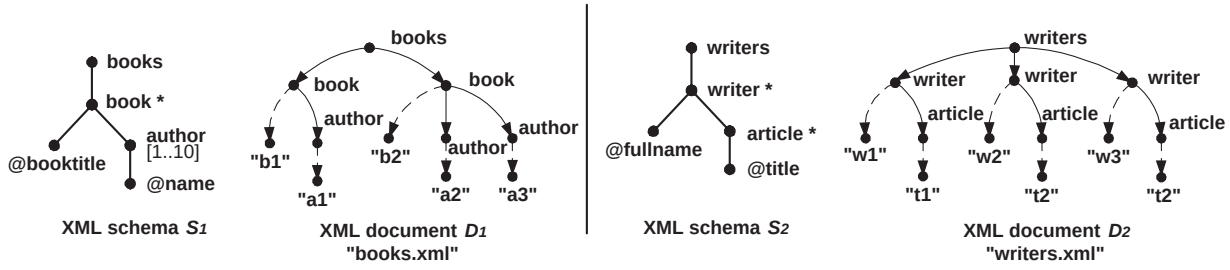

Figure 1: An example of structural heterogeneity between XML documents.

Example 1.1 *Figure 1 gives two XML schemas (S_1 and S_2) with their documents (D_1 and D_2), which are represented as trees. It is obvious that S_1 and S_2 both represent a many-to-many relationship between two concepts:* `book` *and* `author` *(equivalently denoted by* `article` *and* `writer` *in S_2). However, structurally speaking, they are different: S_1 (book-centric schema) has the* `author` *element nested under the* `book` *element, whereas S_2 (author-centric schema) has the* `article` *element nested under the* `writer` *element. Suppose our query target is "Give all the authors of the written work b_2." The XML path expressions that are used to define the search patterns in the two schema trees can be respectively written as* `/books/book/author/@name[/books/book/@booktitle="b2"]` *and* `/writers/writer/@fullname[/writers/writer/article/@title="b2"]`*, where the contents in the square brackets specify the constraints for the search patterns. We notice that although the above two search patterns refer to conceptually equivalent concepts, they follow two distinct XML paths.*

1.2 Semantic integration of XML documents

The diversity of the XML queries (referred to in this paper as *structural queries*) results from the diversity of possible XML schemas (also called *structural schemas*) for a single conceptual model. In comparison, the schema languages that operate on the conceptual level (called *conceptual schemas*) are *structurally flat* so that the user can formulate a determined query (called *conceptual query*) without considering the structure of the source. RDF Schema (RDFS) [16], DAML+OIL, and OWL are examples of languages used to create conceptual schemas. There are currently many attempts to use *conceptual schemas* [1, 2, 10] or *conceptual queries* [8, 9] to overcome the problem of structural heterogeneities among XML sources.

In this paper, we propose an approach to integrate XML sources and handle queries in the integrated system by using a *bidirectional* query translation algorithm. We choose to use the global-as-view (GaV) approach [7] for the integration of XML sources (modeled by XML Schema [11] in our approach). In particular, we first transform the heterogeneous XML sources into local RDF ontologies (defined using RDFS space [6]), which are then merged into the RDF global ontology. This transformation process encodes the mapping information between each concept in the local RDF ontology and the path to the corresponding element in the XML source. The ontology merging process is semi-automatically performed by utilizing the PROMPT algorithm proposed in [17]. Apart from the global ontology, the merging process also produces a *mapping table*, which contains mapping information between concepts in the global ontology and concepts in the local RDF ontologies. In our approach, we can translate a query posed against the global ontology into sub-queries over the sources. We can also translate a query posed against an XML source to an equivalent query against any other XML source. Given that we choose a GaV approach, the global ontology is a view over the local ontologies, therefore the process of mapping a query over the global ontology to queries over

the local ontologies is straightforward.

1.3 Contributions

In brief, we make the following contributions in this paper.

- We propose an approach for using an ontology-based mediation architecture to integrate heterogeneous XML sources. The mediation integrates both the XML nesting structure and the domain structure expressed by RDFS to enable semantic interoperation between the XML sources by hiding their structural heterogeneities. This integration process is lossless with respect to the nesting structure of the XML document.
- We extend the power of RDFS by defining additional metadata. These metadata encode the nested structure of the XML Schema in the RDF schema.
- Finally, we describe an algorithm for translating the query back and forth between XQuery and RDQL. The translation is based on the mapping table and the principle of preserving the nesting structure on the XML sources.

The paper is organized as follows. Section 2 describes related work. Section 3 describes the architecture of our approach. The two key points in our approach, i.e., data integration process and query processing, are discussed respectively in Sections 4 and 5. We draw conclusions and discuss future work in Section 6.

2 Related Work

There are a number of approaches addressing the problem of data integration or interoperability among XML sources. The approaches proposed can be classified into the following three categories.

2.1 Semantic integration

High-level Mediator Amann *et al.* propose an ontology-based approach to the integration of heterogeneous XML Web resources in the C-Web project [1, 2]. The proposed approach is very similar to our approach except for the following differences. The first difference is that they use the local-as-view (LaV) approach [7] with a hypothetical global ontology that may be incomplete. The second difference is that they do not retain the XML documents' structures in their conceptual mediator so they cannot deal with the reverse query translation (from the XML sources to the mediator). Our previous work [10] involved a layered approach for the interoperability of heterogeneous web sources, but the nesting structure associated with XML was lost in the mapping from XML data to RDF data.

Direct Translation Klein proposes a procedure to transform XML data directly into RDF data by annotating the XML documents via external RDFS specifications [13]. The procedure makes the data in XML documents available for the Semantic Web. However, since the proposed approach does not consider the document structure of XML sources, it can not propagate queries from one XML source to another XML source.

Encoding Semantics The Yin/Yang Web [18] proposed by Patel-Schneider and Siméon address the problem of incorporating the XML and RDF paradigms. They develop an integrated model for XML and

RDF by integrating the semantics and inferencing rules of RDF into XML, so that XML querying can benefit from their RDF *reasoner*. But the Yin/Yang Web does not solve the essential problem of query answering across heterogeneous sources, i.e., with different syntax or data models. It also could not process higher-level queries such as RDQL. Lakshmanan and Sadri also propose an infrastructure for interoperating over XML data sources by semantically marking up the information contents of data sources using application-specific common vocabularies [15]. However, the proposed approach relies on the availability of an application-specific standard ontology that serves as the global schema. This global schema contains much information necessary for interoperability, such as key and cardinality information for predicates. This approach has the same problem as the Yin/Yang Web, that is, higher-level queries can not be processed downwards to XML queries.

2.2 Query languages

CXQuery [9] is a new XML query language proposed by Chen and Revesz, which borrows features from both SQL and other XML query languages. It overcomes the limitations of the XQuery language by allowing the user to define views, explicitly specify the schema of the query answers, and query through multiple XML documents. However, CXQuery does not solve the issue of structural heterogeneities among XML sources. The user has to be familiar with the document structure of each XML source to formulate queries. Heuser *et al.* also present a new language (CXPath) based on *XPath* for querying XML sources at the conceptual level [8]. The user can use CXPath to write queries over a conceptual schema that abstracts the semantic content of several XML sources. However, they do not consider the situation of query translation from the XML sources to the global conceptual schema.

2.3 Query rewriting

Query rewriting or query translation is the key issue for both mediator-based integration systems and peer-to-peer systems. As an example of the first case, the Clio approach [19] mainly addresses schema mapping and data transformation between nested schemas and/or relational databases. It focuses on how to take advantage of schema semantics to generate the consistent translations from source to target by considering the constraints and structure of the target schema. The approach uses queries to express the mapping so as to transform the data into the target schema. The Piazza system [12] is a peer-to-peer system that aims to solve the problem of data interoperation between XML and RDF. The system achieves its interoperability in a low-level (syntactic) way, i.e., through the interoperability of XML and the XML serialization of RDF.

3 Architecture

The architecture of our proposed framework is shown in Figure 2. In this section, we discuss the architecture from two aspects: the integration of XML sources into the global ontology and the query processing within the integration system.

3.1 Ontology integration and mapping table

The *ontology integration* process contains two steps: *schema transformation* and *ontology merging*. In the first step, we use RDFS to model each XML source as a local RDF ontology to achieve a uniform representation basis for the ontology merging step. The key operation is the preservation of the nesting

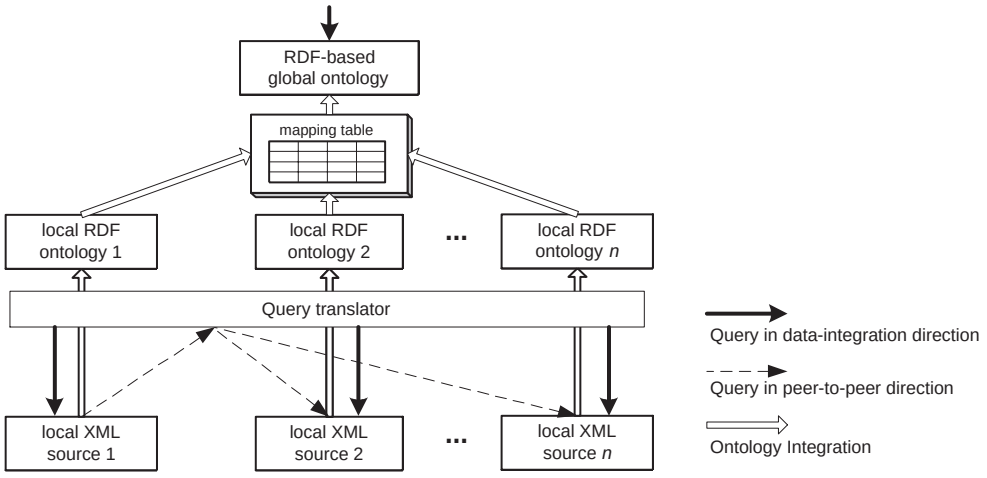

Figure 2: Architecture of the integration framework.

structure of the XML documents. In the second step, we merge all the local RDF schemas to generate the global ontology. In this process, additional domain-related knowledge (e.g., inheritance) may be introduced. During the merging process, a *mapping table* is produced to contain the mapping information between the global RDF ontology and local RDF ontologies. We discuss this process in more detail in Section 4.

It is worth mentioning that the global ontology in our system has two roles: (1) It provides the user accessing the data through the global ontology with a uniform query interface to facilitate a formulation of a single query on all the XML sources; (2) It serves as the mediation mechanism for accessing the distributed data through any of the XML sources.

3.2 Query processing and reversibility

As shown in Figure 2, the translation of queries in our system may occur in two directions:

Data integration The system translates an RDF query (directed to the global ontology) into multiple sub-queries (one for each XML source).

Peer-to-peer integration Query translation is performed in a similar way to query processing in peer-to-peer systems [20]. That is, the query posed by a user against any XML source is propagated to all the mediated distributed sources.

We use XQuery [4] to write queries over the XML sources and we use RDQL (RDF Data Query Language, based on SquishQL [14]), to write queries over the RDF global ontology.

The query translation between structural queries in XQuery and conceptual queries in RDQL involves the problem of *reversibility*. In this subsection, we introduce the concept of reversibility that forms the basis for query translation across syntactically, schematically, and semantically heterogeneous data sources.

We start by looking at the problem for a single source. Intuitively, the query translation mechanism from a source query Q to the target query Q' is *reversible* when we can translate Q' to a query Q'', that produces an answer that is semantically equivalent to that produced by Q on the data source. The left side of Figure 3 illustrates this case.

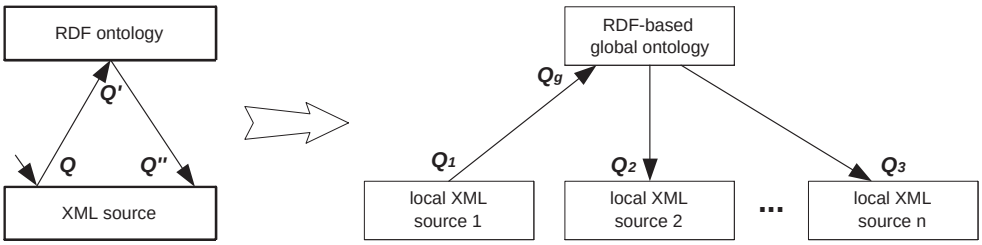

Figure 3: Query reversibility and query translation.

For distributed sources, the user poses a query Q_1 on the *local XML source 1*, which is translated first to Q_g on the global ontology. Then Q_g is translated into multiple sub-queries, Q_2, Q_3, ..., Q_n on the distributed XML sources.

In this case, the query translation is *reversible* when we can translate Q_g into a query Q_i, that produces an answer on the *local XML source i* that is semantically equivalent to the one that would be produced by Q_1 on the *local XML source 1*. Notice that it is our assumption that the n data sources are semantically equivalent, however structurally different. It makes no difference to the notion of reversibility whether the queries are against data with different data models (e.g., RDF and XML), or whether the queries are against the same data source or different data sources as long as they are all semantically equivalent. In Section 5, we use a concrete example (Example 5.3) that illustrates this concept.

4 Integrating Structure and Semantics: A Case Study

To support the reversible query translation process between the structural query (XQuery) and the semantic query (RDQL), we choose to extend the vocabulary of RDF to make it capable of representing not only the semantics but also the structure of the data. In particular, we define a new RDF property *rdfx:contain* (*rdfx* stands for the namespace where the *contain* is defined) to enable RDF representation of the XML nesting structure. The data integration process consists of two sub-processes, *schema transformation* and *ontology merging*, which are discussed respectively in Section 4.1 and Section 4.2 through a case study. In the case study we integrate the two XML schemas S_1 and S_2 (see Figure 1). Figure 4 shows the RDF ontologies (S_1' and S_2') that result respectively from the transformation of S_1 and S_2.

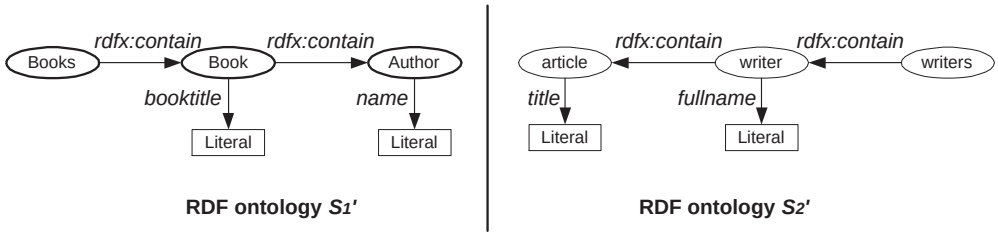

Figure 4: RDF ontologies transformed from XML schemas in the case study.

4.1 Local RDF ontology

In the first sub-process, we transform the local XML schema into a local RDF ontology while preserving the XML document structure. By *document structure*, we mean the structural relationship of objects specified in *data-centric* documents [5] by a schema language (such as DTD, XML Schema, or RelaxNG). In this paper, we focus on the nesting structure (i.e., hierarchy). Other structural properties include *order*. A consequence of not including order in our framework is that we cannot consider a query Q_1 that involves the order of the subelements of an element. However, this kind of query is not of interest in a framework where we are concerned with the semantics of the data.

In terms of XML Schema, elements and attributes are the two basic building blocks of XML documents. Elements can be defined as *simple types* which cannot have element content and cannot carry attributes or *complex types* which allow elements in their content and/or contain attributes. On the other hand, all attribute declarations must reference simple types since attributes cannot contain other elements or other attributes. From the perspective of XML Schema, these nesting relationships are defined in terms of *datatypes* (simple or complex type). A well-formed XML document contains the hierarchical structure of elements and attributes, which contains the following two aspects:

Element and attribute relationship Only complex-type elements can carry attributes and attributes can only be of simple types.

Element and sub-element relationship Likewise, only complex-type elements can allow elements as their children. But child elements can be either simple types or complex types. In this case, there exist two other document structures: order and cardinality, which we will further discuss in the following subsections.

Taking into account XML elements, attributes and their relationships, the transformation from XML to RDF can further include *element-level* transformation and *structure-level* transformation.

Element-level transformation The element-level transformation defines the basic classes and properties of the local RDF ontology according to the transformation correspondences shown in Table 1, with the structural relationships between the elements not being considered for the time being. No new RDF metadata needs to be defined here because *rdfs:Class* and *rdfs:Property* are enough for the specifications of classes and properties. For instance, in the case study, for S_1 we define the classes: `Books`, `Book`, and `Author` while taking `booktitle` and `name` as properties of `Book` and `Author`, respectively (See Figure 4).

Table 1: Correspondences of element-level transformation

XML Schema concepts	**RDF Schema concepts**
Attribute	Property
Simple-type element	Property
Complex-type element	Class

Structure-level transformation The structure-level transformation encodes the hierarchical structures of the XML schema into the local RDF ontology. The encoding involves two relationships: *element-attribute* relationship and *element-subelement* relationship. Following the element-level transformation, it is natural to encode the element-attribute relationship as a *class-to-literal* relationship, and the

element-subelement relationship as a *class-to-class* relationship in RDFS. We define a new RDFS predicates *rdfx:contain* to represent the class-to-class relationships. Specifically, we add a new property with its domain being one class (converted from the parent element), its range being the other class (converted from the subelement), and its name being *rdfx:contain*. As a result, we can see in Figure 4 that *rdfx:contain* enables the representation of the nesting relationship. For instance, by following the edges of *rdfx:contain* from `Books` to `Author` in S_1', we in fact get the corresponding XPath `/books/book/author` in S_1. Table 2 lists the mapping information between the XML source S_1 and the local RDF ontoloy S_1'.

Table 2: Mappings between S_1 and S_1'

XPath expressions in S_1	**RDF expressions in S_1'**
/books	Books
/books/book	Book
/books/book/@booktitle	Book.booktitle
/books/book/author	Author
/books/book/author/@name	Author.name

4.2 Global RDF ontology

The process of ontology merging takes multiple local ontologies (encoded in RDFS) as the input and returns a merged ontology as the output [21]. Ontology merging and ontology alignment are widely pursued research topics. In this paper we do not intend to introduce a new technique for ontology merging. Instead, we utilize existing techniques to generate the integrated ontology from the local ontologies. In particular, we utilize and develop the PROMPT approach [17] so that basic operations of the ontology merging process include:

- *merging of classes*: Merging of multiple conceptually equivalent classes into one class.
- *merging properties*: Merging of multiple conceptually equivalent properties of a class into one property.
- *merging relationships between classes*: Merging of conceptually equivalent relationships from one class c_1 to another class c_2 into one relationship (i.e., an RDF property taking c_1 as its domain and c_2 as its range).
- *copying a class and/or its properties*: Copying a class with its properties directly if the same or equivalent class/property does not exist in the target ontology.
- *generalizing related classes into a superclass*: Taking multiple conceptually related classes as subclasses of a more general class. The superclass can be obtained by searching a existing knowledge domain (e.g., the DAML Ontology Library) or reasoning over a thesaurus.

Figure 5 gives the ontology that results from merging two local RDF ontologies (see Figure 4) in the case study. The grayed classes and properties are merged classes and properties from the original ontologies. For instance, the class `Book` is merged from `Book` in S_1' and `Article` in S_2', whereas the property `title` is merged from `booktitle` in S_1' and `title` in S_2'. The classes `Book` and `Author` are also respectively extended with the superclasses `Publication` and `Person`.

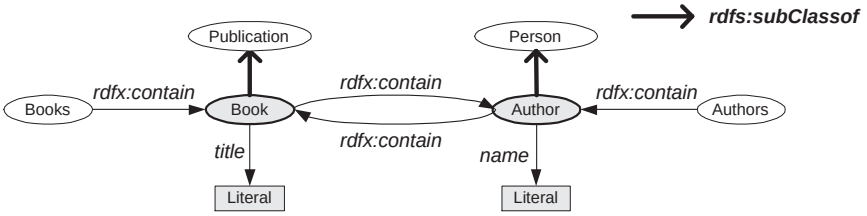

Figure 5: The global ontology merged from local RDF ontologies.

Apart from the global ontology, the ontology merging sub-process also yields another output: the mapping table that contains the mapping information between the local RDF ontologies and the global RDF ontology. In general, if a class, property, or relationship between classes p in the global ontology is the result of merging p_i and p_j from different local ontologies, then a mapping in the form of (p, p_i, p_j) is generated. If a class or property p in the global ontology is only copied from p_i in a local ontology, then a mapping (p, p_i) is produced. For instance, for the class `Book.title` (in the global ontology) that is merged from `Book.booktitle` in S_1' and `Article.title` in S_2', we generate an entry in the mapping table: `(Book.title, Book.booktitle, Article.title)`. Table 3 lists all the mappings in our case study.

Table 3: Mapping table between the global ontology and local RDF ontologies

RDF expressions in the global ontology	**RDF expressions in S_1'**	**RDF expressions in S_2'**
Books	Books	-
Book	Book	Article
Book.title	Book.booktitle	Article.title
Authors	-	Writers
Author	Author	Writer
Author.name	Author.name	Writer.fullname

5 Query Processing

The system can process queries in two directions: the *data-integration* direction, the query on the global ontology is rewritten into subqueries over multiple sources, and the *peer-to-peer* direction, the query on some XML source is propagated to the XML sources connected through the global ontology. In this section, we describe the query rewriting algorithm by using concrete examples.

5.1 Assumptions

To represent queries, RDQL uses an SQL-like syntax, in which the SELECT clause identifies the variables to be returned to the application. The FROM clause specifies the RDF model using an URI. The WHERE clause specifies the graph pattern as a list of triple patterns. The AND clause specifies the Boolean expressions. Finally, the USING clause provides a way to shorten the length of the URIs. Whereas XQuery is a functional programming language which has an FLWR (i.e., for, let, where, return) syntax. For simplification, we make the following assumptions for the query rewriting algorithm.

- We assume that the XML sources to be integrated are related to the same knowledge domain, and data instances are well populated in these sources so that we do not necessarily consider handling *null* values from the XML sources.
- We also assume that the XML query posed by the user is formulated only in the form of *FLWR expressions* [4]. However, we are not going to discuss translation of nesting XML queries in the rewriting algorithm, although nesting FLWR expressions are allowed in XQuery.
- We finally assume that all the concepts in every local ontologies are mapped to the concepts in the global ontology during the ontology integration process. That is, the mappings are total, one-to-one mappings from the local RDF ontologies to the global ontology. It is possible that some concept, say c_i, in the global ontology gets mapped to a local ontology but not to another local ontology. This leads to the problem of *null* values when a query involves c_i.

5.2 Data integration

The global ontology connects all the individual local RDF ontologies that represent local XML sources and provides a uniform query interface for the user. Thus, the user can retrieve data from all the sources in the system by simply submitting a single RDF query on the global ontology. In this subsection, we discuss translation from the RDQL query on the global ontology to the XQuery queries over the XML sources. In the following discussion, we use M to denote the mapping table (between the global ontology and the local RDF ontologies), Q_g for the input RDQL query (on the global ontology), Q_r for the intermediate query (on the local RDF ontology), and Q_x for the output XQuery query (on the XML source). The process of query rewriting from RDQL to XQuery proceeds as follows:

Step 1 Identify the RDF path expression corresponding to each variable that is used in Q_g, put these RDF path expressions into a set P, and group them into different sets based on their different roles: (1) Put the RDF expression that appears in the SELECT clause into a set P_s. (2) Put the RDF expression, which appears in the WHERE clause and is constrained with a *Literal* constant or a URI, into a set P_w. (3) Also put the RDF expression that occurs in the AND clause into P_w.

Step 2 For each participating local RDF ontology R_i, based on the correspondences in the mapping table M, replace the RDF paths in P with the RDF paths mapped in R_i, while updating P_s and P_w in the same way. Rewrite Q_g into Q_r by using the following method: (1) For the WHERE clause, traverse R_i to find an acyclic sub-graph that covers all the path expressions in P, and put all the edges (properties) into a set E. For each $e_i \in E$ (with the subject s_i and the object o_i) , add a triple in the form of ($?s_i$, e_i, $?o_i$) into the WHERE clause. (2) For the SELECT and AND clauses, simply replace the RDF paths in them with their corresponding paths in R_i, then bind each path with a variable.

Step 3 Find the XPath expression corresponding to every element p_i in both P_s and P_w, by utilizing the mapping information between the local XML source and its local RDF ontology R_i. For the result, we use P'_s to denote the set of XPaths corresponding to P_s, and P'_w for P_w.

Step 4 Construct the target query Q_x for each XML source according to the following rules:

- **The let Clause:** Output `let $<`*root label*`> := doc("<`*XML source name*`>")`.
- **The for Clause:** For each XPath p_i in both P'_w and P'_s, output a for clause in the form of `for $<`*node label*`> in <` p_i `>`.

- **The where Clause:** For each p'_i in P'_w, construct a query condition according to the constraints in Q_r, and take the conjunction of these conditions as the where clause.
- **The return Clause:** Take each p'_i in the set P'_s as an element in the return clause.

Step 5 Abbreviate each absolute XPath p_i referred in Q_x into a relative XPath by replacing some part of p_i with v, if this part was bound to v.

Example 5.1 *Suppose we submit against the global ontology in our case study a query Q_g: "List all the book titles.", whose RDQL-syntax code is shown below. We assume that the global ontology is defined in the namespace: http://examples.org/globalontology# and the local ontology R_i is defined in the namespace: http://examples.org/localontology-i# (we will be making the same assumptions in all the examples of this section). In this example, we illustrate the algorithm by translating Q_g into a query on* `books.xml` *(see Figure 1).*

```
SELECT ?title
WHERE (?book, <go:title>, ?title)
USING go for <http://examples.org/globalontology#>
```

After Step 1, we get P = {`Book, Book.title`}, P_s = {`Book.title`}, and P_w is empty. By looking into the mapping information in Table 3, we update P = {`Book, Book.booktitle`} and P_s = {`Book.booktitle`}, and rewrite Q_g into Q_r as shown below (after Step 2).

```
SELECT ?booktitle
WHERE (?book, <lo:booktitle>, ?booktitle)
USING lo for <http://examples.org/localontology-1#>
```

In Step 3, we find that `/books/book/@booktitle` in `books.xml` is the corresponding XPath of `Book.booktitle` in the local ontology R_1. Thus, P'_s = {`/books/book/@booktitle`} and P'_w is still empty. By following the XQuery construction instructions in Step 4 and XPath abbreviation method in Step 5, we output the target query Q_x as:

```
let $books := doc("books.xml")
for $booktitle in $books/book/@booktitle
return $booktitle
```

Example 5.2 *Suppose a query Q_g "List all the books written by a_2." is posed by the user over the global ontology. We are supposed to translate it into a query Q_x against* `writers.xml`*.*

```
SELECT ?title
WHERE (?book, <go:title>, ?title),
      (?book, <rdfx:contain>, ?author),
      (?author, <go:name>, ?name)
AND (?name eq "a2")
USING go for <http://examples.org/globalontology#>
```

After Step 1, we get P = {`Book, Book.title, Author, Author.name`}, P_s = {`Book.title`}, and P_w = {`Author.name`}. In Step 2, by looking into the mapping table M, we update P = {`Article, Article.title, Writer, Writer.fullname`}, P_s = {`Article.title`}, P_w = {`Writer.fullname`}, and rewrite Q_g into Q_r as shown below.

```
SELECT ?title
WHERE (?article, <lo:title>, ?title),
      (?article, <rdfx:contain>, ?writer),
      (?writer, <lo:fullname>, ?fullname)
AND (?fullname eq "a2")
USING lo for <http://examples.org/localontology-2#>
```

In Step 3, we get $P'_s = \{$`/writers/writer/@fullname`$\}$ and $P'_w = \{$`/writers/writer/article /@title`$\}$. By following the XQuery construction instructions in Step 4 and XPath abbreviation method in Step 5, the resultant target query Q_x as follows:

```
let $writers := doc("writers.xml")
for $writer in $writers/writer
      for $title in $writer/article/@title
where $writer/@fullname = "a2"
return $title
```

5.3 Peer-to-peer integration

The query rewriting in the peer-to-peer direction contains three phases: (1) The input XQuery query Q_x posed by the user against an XML source is translated into an RDQL query Q_r over its corresponding local RDF ontology. (2) Q_r is translated into an equivalent RDQL query Q'_r over each peer local RDF ontology by using the mapping information in the mapping table (See Table 2). (3) Q'_r is rewritten into the query Q'_x over the XML source. It is obvious that the last two phases proceed in the same way as the query rewriting in the *data-integration* direction. The first phase consists of the following steps:

Step 1' Extend the relative XPaths referred to in the Where clause and in the Return clause of the original XML query respectively into a set of absolute XPaths, denoted by P_1 for the Where clause and by P_2 for the Return clause.

Step 2' Based on the mapping information between the local XML source and its RDF ontology R_i, we can get the RDF path expressions corresponding to the XPaths in both P_1 and P_2, using P'_1 and P'_2 respectively to denote the resulting sets of RDF paths for P_1 and P_2.

Step 3' Construct the query Q_r against R_i according to the following rules.

- **The SELECT Clause:** Bind each RDF path p_i in P'_2 to a variable v_i, whose name is the label of p_i in the RDF ontology. Add each v_i into the SELECT clause as an item and output: `SELECT ?<`v_1`> , ... , ?<`v_i`> , ... , ?<`v_n`>`.
- **The WHERE Clause:** Traverse R_i to find an acyclic sub-graph that covers all the RDF expressions in both P'_1 and P'_2, and put all the edges (properties) into the set E. For each e_i (with s_i as its subject and o_i as its object) in E, add a triple in the form of ($?s_i$, e_i, $?o_i$) into the WHERE clause.
- **The AND Clause:** For each p_i in P'_1, construct a query condition according to the original constraints in Q_x. Take the conjunction of these conditions as the AND clause.

- **The FROM and USING Clause:** These clauses are related to URIs and therefore are not further discussed.

Example 5.3 *Suppose the user posed the original query Q_x: "List all the authors." on the XML source* `writers.xml`*. Our algorithm will rewrite it into a query on the other XML source* `books.xml`*. The XQuery for the original query can be written as follows.*

```
let $writers:= doc("writers.xml")
for $writer in $writers/writer
return $writer
```

After Step 1, we obtain an empty P_1 and $P_2 = \{$`/writers/writers`$\}$. After Step 2, P_1' is empty and we have $P_2' = \{$`Writer`$\}$. Following the instructions in Step 3, we can construct the RDQL query Q_r that only refers to the class `Writer` in the local ontology S_2' (See Figure 4). By simply applying the rewriting algorithm from RDQL to XQuery on Q_r, we get the final resulting query Q_x' (against `books.xml`):

```
let $books := doc("books.xml")
for $author in $books/book/author
return $author
```

As discussed in Section 3.2, the query translation from XQuery through RDQL to XQuery involves the concept of reversibility. We have shown that the translation reversibility depends on the equivalence between the source query (Q_x in Example 5.3) and the target query (Q_x' in Example 5.3), which depends on the equivalence of the answer sets to Q_x and to Q_x'. In Example 5.3, the answer sets to Q_x and Q_x' are shown below, where we can see that the two answer sets are structurally different but semantically equivalent.

```
<author name="a1"/>
<author name="a2"/>
<author name="a3"/>
```

Fragment of the answer set to Q_x.

```
<writer fullname="w1">
    <article title="t1"/>
<writer>
...
```

Fragment of the answer set to Q_x'.

6 Conclusions and Future Work

XML and its schema languages do not express semantics but rather the document structure, such as the information about nesting. Therefore, semantically-equivalent documents often present different document structures that originated in various applications. In this paper, we provide an ontology-based framework that aims to make two XML documents interoperate at the semantic level while retaining their nesting structure. In our approach, an RDF-based global ontology is generated by merging the RDF ontologies that are generated from each of the XML documents. We extend RDF Schemas by defining additional metadata that can encode the nesting structure of an XML document. We propose two query rewriting algorithms: one that translates an RDF query (posed on the global ontology) to an XML underlying database and the other one that translates an XML query (posed on one of the individual databases) to an RDF query on the ontology, followed by the translation to an XML query (on one of the individual databases). The last query

processing mechanism is therefore *bidirectional* and corresponds to a P2P mode of operation. We introduce the concept of *reversibility* to guarantee the correctness of the bidirectional query processing.

In future work we will: (1) Prove the correctness of our query translation algorithms for a large subset of the XQuery language. (2) Propose an approach for the unification of the results (as expressed in XML) with different structure or representation, which are returned from different sources.

References

[1] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *Proceedings of the 1st International Semantic Web Conference (ISWC 2002)*, pages 117–131, 2002.

[2] B. Amann, I. Fundulaki, M. Scholl, C. Beeri, and A. Vercoustre. Mapping XML Fragments to Community Web Ontologies. In *Proceedings of the 4th International Workshop on the Web and Databases (WebDB 2001)*, pages 97–102, 2001.

[3] Y. Bishr. Overcoming the semantic and other barriers to GIS interoperability. *International Journal of Geographical Information Science*, 12(4):229–314, 1998.

[4] S. Boag, D. Chamberlin, M. F. Fernández, J. R. D. Florescu, and J. Siméon. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery, W3C Working Draft, August 2003.

[5] R. Bourret. XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[6] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema, W3C Working Draft, January 2003.

[7] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the Expressive Power of Data Integration Systems. In *21st Intl. Conference on Conceptual Modeling (ER)*, pages 338–350, 2002.

[8] S. D. Camillo, C. A. Heuser, and R. S. Mello. Querying Heterogeneous XML Sources through a Conceptual Schema. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER2003)*, pages 186–199, 2003.

[9] Y. Chen and P. Revesz. CXQuery: A Novel XML Query Language. In *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Science, and Medicine on the Internet (SSGRR 2002w)*, 2002.

[10] I. F. Cruz and H. Xiao. Using a Layered Approach for Interoperability on the Semantic Web. In *Fourth International Conference on Web Information Systems Engineering (WISE'03)*, pages 221–232, Roma, Italy, December 2003.

[11] D. C. Fallside. XML Schema Part 0: Primer. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[12] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *Proceedings of the 12th International World Wide Web Conference (WWW2003)*, pages 556–567, 2003.

[13] HP Labs. RDQL - RDF Data Query Language. http://www.hpl.hp.com/semweb/rdql.htm.

[14] M. C. A. Klein. Interpreting XML Documents via an RDF Schema Ontology. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA 2002)*, pages 889–894, 2002.

[15] L. V. Lakshmanan and F. Sadri. Interoperability on XML Data. In *Proceedings of the 2nd International Semantic Web Conference (ICSW'03)*, 2003.

[16] F. Manola and E. Miller. RDF Primer. http://www.w3.org/TR/rdf-primer, W3C Working Draft, January 2003.

[17] F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2000)*, pages 450–455, 2000.

[18] P. F. Patel-Schneider and J. Siméon. The Yin/Yang web: XML syntax and RDF semantics. In *Proceedings of the 11th International World Wide Web Conference (WWW2002)*, pages 443–453, 2002.

[19] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating web data. In *Proceedings of VLDB*, pages 598–609, 2002.

[20] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. Query Processing Over Peer-To-Peer Data Sharing Systems. Technical Report CSD-2002-28, University of California at Santa Barbara, 2002.

[21] G. Stumme and A. Maedche. Ontology Merging for Federated Ontologies on the Semantic Web. In *Proceedings of the International Workshop for Foundations of Models for Information Integration (FMII-2001)*, pages 413–418, 2001.