Integrating Heterogeneous Data Source Using Ontology

Jinpeng Wang, Jianjiang Lu, Yafei Zhang, Zhuang Miao and Bo Zhou
Institute of Command Automation, PLA University of Science and Technology, Nanjing, China
Email: wangjinpeng1982@gmail.com

***Abstract*—Integrating data from multiple heterogeneous sources entail dealing with different data models, schemas and query languages. The burgeoning Semantic Web has provided several new methods for data integration. This paper focuses on integration of relational database and XML data. To solve the problem we propose an ontology-based approach. A semantic integration infrastructure for heterogeneous data sources is presented. In this infrastructure, ontology is used as the mediated schema for the representation of the data source semantics. To model different source schemas, we propose a describing method based on RDF graph patterns. The semantic mappings between source schema and RDF ontology are described declaratively using SPARQL queries. The semantic of query rewriting is further discussed and a query rewriting algorithm is presented.**

***Index Terms*—data integration, ontology, SPARQL, heterogeneous database, RDF**

I. Introduction

Integrating and querying data from heterogeneous sources is a hot research topic in database research field. The goal of data integration is to provide user a uniform access to multiple heterogeneous data sources. This problem is known in the literature as query rewriting and query answering using views, and has been studied very actively in the recent years [1]. However, with the use of ontology, these former research works are not applicable.

The burgeoning Semantic Web has provided several methods for integration of heterogeneous data sources. Being an "explicit specification of a conceptualization" [1], ontology is considered as a possible solution to represent the content of heterogeneous data sources. RDF is a general proposition language for description of ontology. SPARQL, a query language for RDF, can join data from different databases, as well as documents, inference engines, or anything else that might express its knowledge as a directed labeled graph.

In this paper, an ontology-based approach for heterogeneous data source integration is proposed. In our proposal, RDF ontology is used as mediated schema for the explicit description of the data source semantics, providing a shared vocabulary for the specification of the semantics. In order to implement the integration, the meaning of the source schemas has to be understood. A source describing method based on RDF graph patterns is proposed to specify the semantic mapping between ontology and the source schemas. The semantic of query rewriting is further discussed and a query rewriting algorithm is presented to reformulate a SPARQL query into source specific queries (e.g. SQL, XQuery etc.), so that SPARQL can access heterogeneous data sources without converting the data into physical triples. We base our discussion on our previous work [3].

The contributions of the present work are:

- A semantic integration infrastructure for relational data.
- A data source describing method based on SPARQL graph patterns.
- Semantic mapping between relational, XML schema and RDF ontology.
- A query rewriting algorithm in integration scenario.

The remainder of this paper is structured as follows: In Section II we first list some related work and Section III overview the integration infrastructure, introduce the architecture of data integration system we present. In Section IV we discuss semantic mapping in detail. After introduce the RDF graph model, relational data model and XML data model, we give some principles for semantic mapping between data source schemas and ontology. Then a new method which describe source schema using SPARQL is proposed. Section V discusses the semantic of query rewriting on graph pattern. In Section VI, we give our algorithm to execute the query rewriting. Section VII concludes the paper.

II. Related Work

A. Relational Database Integration

Considering that much of the world's data are stored in databases of one kind or another, primarily relational databases, need for integrating of heterogeneous relational databases efficiently is obvious. Systems like Virtuoso [4], D2RQ [5] and Squirrel RDF [6] rewrite SPARQL queries to SQL. Oracle's 10.2 and the MySQL SPASQL [7] module compile SPARQL queries directly into an evaluation structure to be executed by the relational engine. However these research works only focused on accessing conventional relational databases using SPARQL, which didn't take the integration into account.

A few early works study integration of relational databases using ontology [8][9], [10]. All the three papers construct ontologies from relational databases schema. This method cannot fully capture the semantic behind the relational schema, only reflecting the structure of data-

doi:10.4304/jsw.4.8.843-850

base, which makes it hard to integrate heterogeneous relational databases.

B. XML Data Integration

The success of XML and its potential use in interoperability problems has fuelled lots of XML-related data integration projects like XQuare [11], Liquid Data (Enosys) [12], Tukwila [13], etc.

Tukwila system uses the MiniCon [14] algorithm to reformulate the queries posed over the global schema into queries over the local XML sources.

Enosys XML Integration Platform is an XML-based data integration system. This system is based on the wrapper-mediator architecture. It allows querying heterogeneous data sources abstracted with XML schemas. Wrappers use XML schemas as logical views of the sources, and a mediator resolves XQuery expressions over the sources.

XQuare (XQuery Advanced Runtime Environment) is a set of open source Java modules for extending J2EE platforms with XML-based, heterogeneous information integration capabilities. The goal of XQuare is presenting to applications a single, uniform XML view of the different data sources, which can then be queried with XQuery to produce XML documents. Accessible data sources include relational databases, XML documents, Web Services and any XQuery-enabled data source and JCA connectors.

III. Integration Architecture

In this section, we discuss the architecture the for data integration. Our approach adopts a so-called mediator-wrapper architecture that allows data sources to function independently while the remote access can be done via a mediator and adaptable wrappers. Illustrated in Figure 1, the architecture of our system may be divided into four layers: application layer communicate with users; mediating layer contains a mediator which allows the integration; wrapper layer contains wrappers for each data resource; and source layer contains a set of heterogeneous sources.

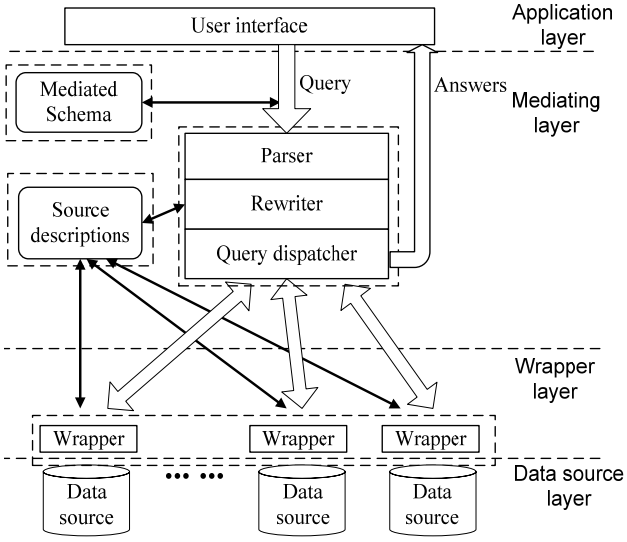


Figure 1. Architecture of data integration system

The relations in the mediated schema are virtual in the sense that their extensions are not actually stored anywhere. The data integration system has a set of source descriptions that specify the semantic mapping between the mediated schema and the source schemas and uses these source descriptions to reformulate a user query into a query over the source schemas.

Circled by broken line, the main elements of the architecture include four parts: query processor, mediated schema, source description and wrapper.

A. Query Processor

The query processor is the kernel component of data integration system. It parses, translates, rewrites and dispatches the user query to related data sources. When user pose their queries expressed in SPARQL using terms from mediated schema, the parser analyzes the query, verifying if it is in accordance with the SPARQL syntax. Rewriter implements the query rewriting algorithm to carry out the query rewriting work with reference to source descriptions.

B. Mediated Schema

We use ontology as the mediated schema, which can be seen as a knowledge base of a particular domain we are interested. The mediated schema has two roles: (1) It provides the user access to the data with a uniform query interface to facilitate the formulation of a query on all sources; (2) It serves as a shared vocabulary set for wrappers to describe the content in every data sources. The mediated schema is expressed using RDFS in our work.

C. Source Descriptions

As mentioned before, queries are posed in terms of the mediated schema. To answer a query, the rewriter need descriptions that relate the contents of each data source to the classes, attributes and relations in the mediated schema. Each data source is described by one or more SPARQL queries. These semantically rich descriptions help the rewriter to form queries and also direct the query dispatcher to distribute queries to specific data sources.

D. Wrapper

The wrapper provides an SPARQL view representing a data source and a means to access and to query the data source. It translates the incoming queries into source-specific queries executable by the query processor of the corresponding sources.

IV. Semantic Mapping

A. RDF Graph Model

RDF describes things by making statements about an entity's properties. RDF statements are triples consisting of a subject (the resource being described), a predicate (the property) and an object (the property value). The values of a statement are URI references. This simple model of assertions leads to a network of information resources, interrelated by properties which establish relations between resources and property values. Thus, one can intuitively understand a collection of information

resources and RDF statements describing them as a graph. To emphasize this characteristic, the term RDF Graph is defined as a set of RDF triples; hence, any collection of RDF data is an RDF Graph [14].

Definition 1 (RDF Terms, Triples, and Variables) Assume there are pair wise disjoint infinite sets *I*, *B*, and *L* (IRIs, Blank nodes, and literals). A tuple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple. In this tuple, *s* is the subject, *p* the predicate and o the object. We denote the union $I \cup B \cup L$ by *T* (RDF terms). Assume additionally the existence of an infinite set *V* of variables disjoint from the above sets.

Definition 2 (RDF Graph) An RDF graph is a set of RDF triples.

Figure 2 shows a part of RDF ontology. We use "sub" represent relation "subclass" for short. There is a relation "write" between concepts "Person" and "Publication", indicating the relationship between authors and their works. The relation "belongs to" between "Person" and "Organization" indicate what organizations people belong to. The relation "per_name" and "per_email" point a person's name and email. The same applies to "pub_title", "pub_year" and "org_name".

B. Relational Model

The term relation is used here in its accepted mathematical sense. Given sets $D_1, D_2, \ldots, D_n$, *R* is a relation on these *n* sets if it is a subset of the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$. *R* is said to have degree *n*, often called *n*-ary. An *n*-ary relation *R* can be represented as a table with *n* column, which has the following properties:

1. Each row represents an *n*-tuple of *R*.
2. The ordering of rows is immaterial.
3. All rows are distinct.

Normally, one column (or combination of column) of a given relation has values with uniquely identify each element (*n*-tuple) of that relation. Such a column (combination) is called a primary key.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. We shall call a column of relation *R* a foreign key if it is not the primary key of *R* but its elements are values of the primary key of some relation *S*.

As shown in Figure 3(a), suppose there are two heterogeneous relational databases, each has several tables containing information about authors and papers.

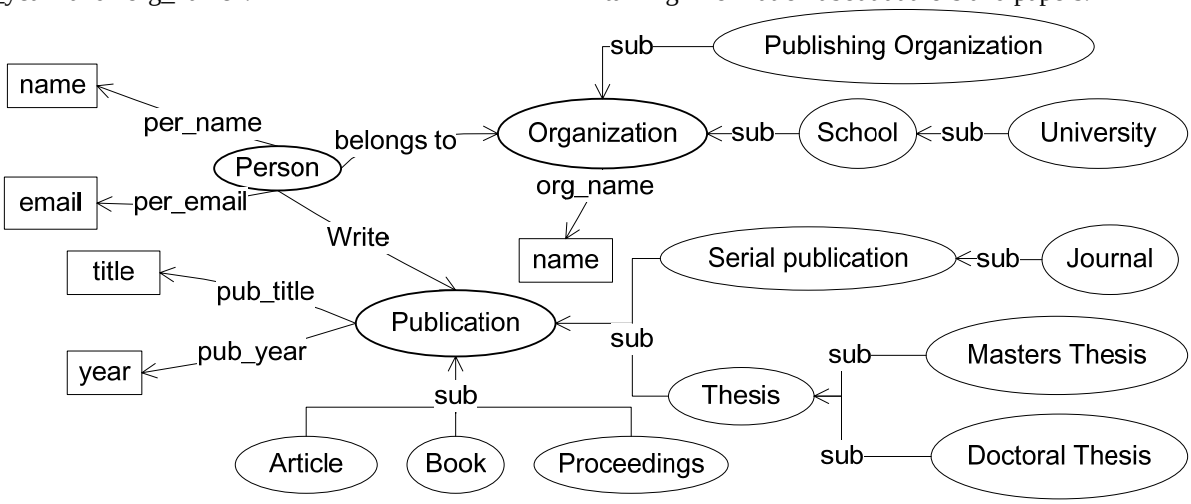
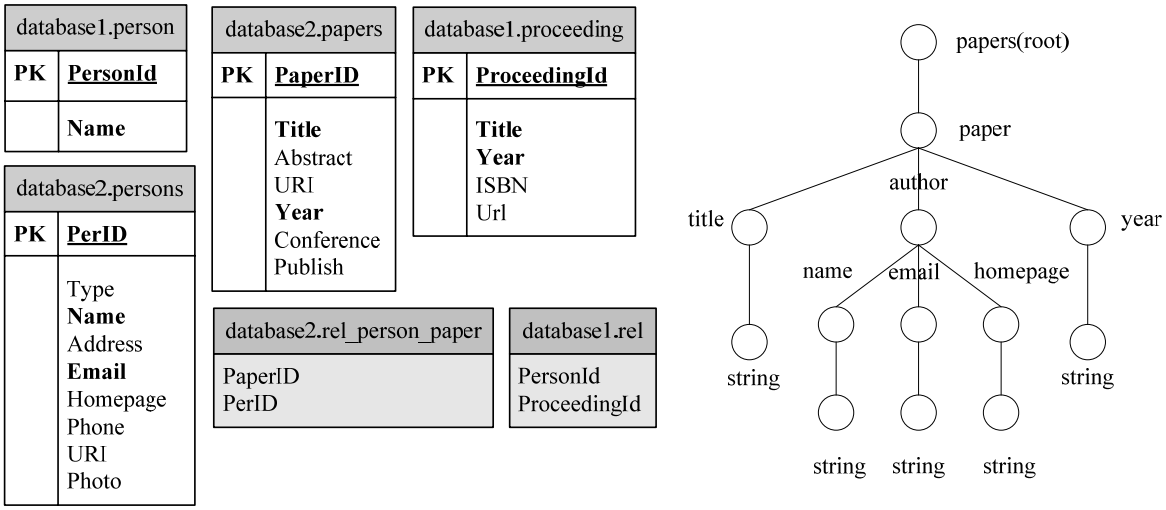


Figure 2. Part of RDF ontology



(a) The relational tables (b) The XML tree

Figure 3. Two heterogeneous data sources

C. XML Model

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. While XML was originally designed to meet the challenges of large-scale electronic publishing, it plays an increasingly important role in the exchange of a wide variety of data on the Web [16].

XML is an extremely versatile data format that has been used to represent many different kinds of data, including web pages, books, XML representations of relational database tables, programming interfaces, objects and multimedia presentations. In addition, some systems offer XML views of non-XML data sources such as relational databases, allowing XML-based processing of data that are not physically stored as XML.

The XML data model is a tree with labeled and ordered nodes. This means that the order of each node is meaningful and important for writing an XML document.

The W3C XML Schema Definition Language [17] is an XML language for describing and constraining the content of XML documents. W3C XML Schema is a W3C Recommendation. The basic characteristic of the XML Schema is that it can define elements and attributes that can appear in a document, define which elements are child elements, the order of these elements and their cardinality. XML Schema provides user with the ability to define an element or an attribute into a specific scope. User can define complex or simple elements (depending on whether they have further structure on not) and cardinalities for them.

Figure 3(b) shows an XML tree, which describing the schema of an XML document about papers and their authors.

D. Mapping Relational Schemas to ontology

To solve the heterogeneity problem, the meaning of the relational schema has to be well described, which is called "source description" in [18]. Before presenting how source descriptions are defined, it is worth considering how, in general terms, relational schema can be mapped to RDF ontology.

As mentioned above, RDF describes resources using a graph model. RDF Schema (RDFS) provides modeling primitives for defining classes and properties, range and domain constraints on properties, and subclass and subproperty relations.

The relational schema is based on entity-relationship diagram (ERD). Typically, each entity is represented as a database table, each attribute of the entity becomes a column in that table, and relationships between entities are indicated by foreign keys. Each table typically defines a particular class of entity, each column one of its attributes. Each row in the table describes an entity instance, uniquely identified by a primary key. The table rows collectively describe an entity set.

There are some similarities and differences between RDF and the ER model, Berners-Lee discussed this issue in [19]. Basically, ERD and RDF Graph have much in common. RDF can be viewed as a member of the Entity-Relationship model family [20]. Class in RDFS corresponds to entity in ERD; property is kind of binary relationship; subclass and subproperty are subsumption relations. Therefore, we list some principles below for semantic mapping between relational schema and RDF ontology:

- Primary key of table is mapped to class.
- Columns in table are mapped to properties.
- Column value is mapped to property value.
- Each row key corresponds to an instance.
- Each row is represented in RDF by a collection of triples with a common subject.

As shown in figure 4, the semantic mappings between relational schemas and ontology are marked by arrows with broken line.

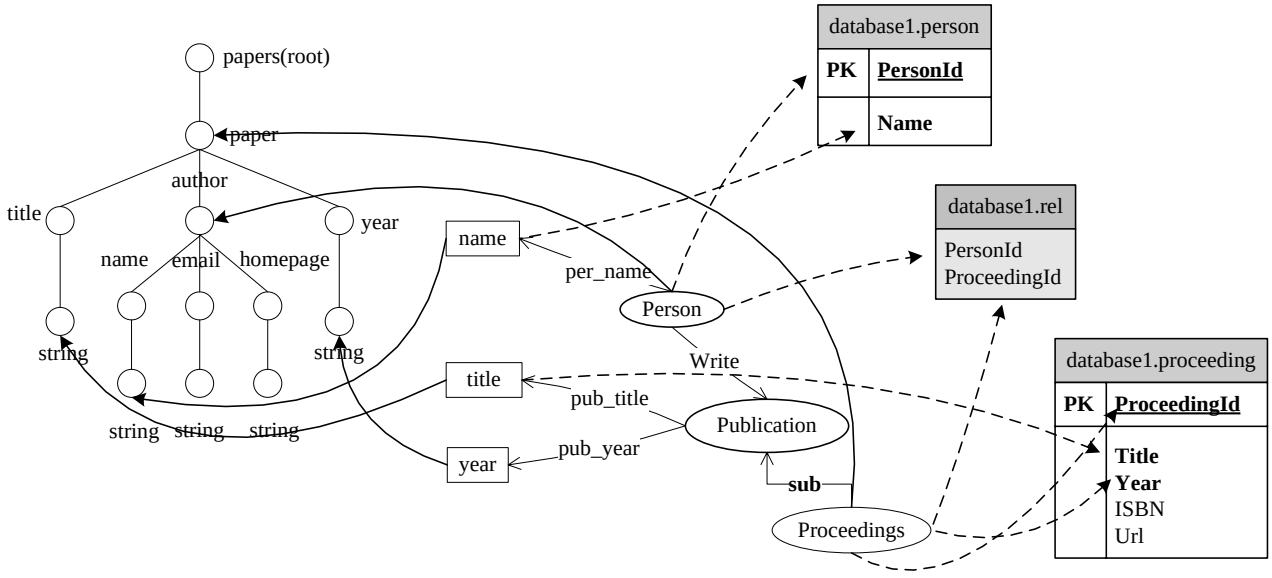


Figure 4. Semantic mapping between relational schemas and ontology

E. Mapping XML Schemas to ontology

Elements and attributes are the two basic building blocks of XML documents. Elements can be defined as simple types or complex types. Simple types cannot have element content and cannot carry attributes, while complex types can contain elements and 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 data types (simple or complex type). A well-formed XML document contains the hierarchical structure of elements and attributes, which contains the following two aspects: only complex-type elements can carry attributes and attributes can only be of simple types; only complex-type elements can allow elements as their children. But child elements can be either simple types or complex types.

No new RDF metadata needs to be defined here because RDFS Class and RDFS Property are enough for the specifications of classes and properties. Taking into account XML elements, attributes and their relationships, we propose some principles below for semantic mapping between XML schema and RDF ontology:

- Attributes is mapped into properties.
- Simple-type elements are mapped into properties.
- Complex-type elements are mapped into classes.

As shown in figure 4, the semantic mappings between XML schemas and ontology are marked by arrows with solid line. In this example, attributes (like title, name, year, etc.) are mapped into properties in ontology. And complex-type elements (like author, paper) are mapped into RDFS classes.

F. Source Description

After being mapped to ontology, source schemas have to be described so that the query rewriting algorithm can use them to generate executable query plans efficiently. We use SPARQL here..

Definition 3 (Source description) Given a data source P, its source description D_P is a tuple (Q_P, μ), where Q_P is a SPARQL query and μ is a mapping from variables appeared in Q_P to corresponding columns in P.

Each relational table is described by a SPARQL query over domain ontology. The semantic of the databases are thus explicitly defined, and it is easy to add and delete sources. Table database1.person can be described as the tuple

```
(
SELECT ?name
WHERE {?X per_name ?name},
{?name→Name}
).
```

Table database1.proceeding can be described as the tuple

```
(
SELECT ?title, ?year
WHERE {?X pub_title ?title.
       ?X pub_year ?year},
{?title→Title, ?year→Year}
).
```

Table database1.rel can be described as a tuple

```
(
SELECT ?person, ?proceeding
WHERE {?person Write ?proceeding}",
{?person→ PersonId,
?preceeding → ProceedingId}
).
```

An XML document can also be described by a SPARQL query. The XML tree in figure 3(b) can be described as the tuple

```
(
SELECT ?title, ?name, ?year
WHERE {?X per_name ?name.
       ?Y pub_title ?title.
       ?Y pub_year ?year.
       ?X Write ?Y},
{?name→/papers/paper/author/name,
 ?title→/papers/paper/title,
 ?year→/papers/paper/year}
).
```

V. SEMANTIC OF QUERY REWRITING

The problem of query rewriting considers how to reformulation a query from the mediated schema to the underlying relational database. A survey and analysis on different algorithms to solve the problem is given in [14]. In this paper, we propose an algorithm to execute the query rewriting over RDF Graph model.

Pérez introduce the formal semantics of SPARQL in [21]. The SPARQL query language is based on matching graph patterns. Graph patterns contain triple patterns that are like RDF triples, but with the option of query variables in place of RDF terms in the subject, predicate or object positions. Combining triple patterns gives a basic graph pattern, more complex graph patterns can be formed by combining smaller patterns.

The triple patterns of a SPARQL query over domain ontology can be considered as a sub-graph of the RDF Graph.

Definition 4 (Query Sub-graph) Given a SPARQL query Q, its Query Sub-graph G_Q is a set of RDF triples that satisfy the query.

In Section IV we define the source descriptions of data source using SPARQL queries. Each SPARQL query corresponds to a sub-graph of the RDF Graph of domain ontology. Consequently, the contents of a data source can be considered as a sub-graph of RDF Graph.

Definition 5 (Source Sub-graph) Given a data source P, its Source Sub-graph G_P is a set of RDF triples that satisfy the queries in source description of the data source.

Given a SPARQL query over domain ontology, query rewriting algorithm finds proper data sources whose subgraph cover the SPARQL query, generates the query plans and unit the results together. The semantic of query rewriting is defined below:

Definition 6 (Query Rewriting) Given a SPARQL query *Q* over domain ontology, a query rewriting answer is a set of data sources whose sub-graph cover the SPARQL query. The results of all possible query rewriting answers of *Q* make up the final result of *Q*.

VI. Query Rewriting Algorithm

As implied by the previous section, a semantically correct query plan amounts to finding proper data sources whose sub-graph cover the SPARQL query. In this section, we present the query rewriting algorithm to generate semantically correct and executable query plans under the RDF Graph model. Our algorithm proceeds in two stages. In the first stage, we find all relative data sources and decompose them to Minimal Connectable Units (MCU) according to source descriptions. In the second stage we join MCUs to produce query plans that are semantically correct and executable.

A. Generating MCUs

A MCU is a subset of RDF triples in source description that can be joined with other MCUs and executed on heterogeneous data sources. Intuitively, a MCU represents a fragment of semantic mapping from the query to the rewriting of the query.

Formally, we define MCUs as follow:

Definition 7 (Minimal Connectable Units) Given a SPARQL query *Q*, a data source *P* and its source description $D_P = (Q_P, \mu)$, a MCU *m* for *P* is a tuple of the form (Y, μ') where:

- Y is a subset of triple patterns in Q_P.
- μ' is a partial mapping from variables appeared in Q_P to corresponding component in *P*.

The algorithm for creating the MCUs is shown in Figure 5. We say a set of triple patterns Y connectable if the following conditions hold: (1) Each return variable of *Q* is also return variable in Y. (2) If x is not a return variable in *Q*, then for every triple pattern t that includes x, $t \in Y$.

Consider a SPARQL query asking for titles of papers written by an author called "Andy":

```
SELECT ?title
WHERE {?X per_name "Andy".
       ?X Write ?Y.
       ?Y title ?title. }
```

```
findMCUs(Q, D_P)
/*Q is a SPARQL query, D_P is the source description of a data source P, has the form (Q_P, μ). */
Initialize M = Φ;
for each triple pattern t in Q, do
for each triple pattern t' in Q_P, do
    if exist a mapping τ that map t to t', then
    find the minimal subset (denoted by Y) of triple patterns of Q_P that is connectable.
    find the subset (denoted by μ') of μ that relative to Y.
    M = M ∪ < Y, μ' >;
end for
end for
return M;
```

Figure 5. Finding the MCUs

TABLE I
MCUs FORMED

No.	*Y*	μ'
1	?x per_name ?y	?y →person.Name
2	?x pub_title ?y	?y →proceeding.Title
3	?x Write ?y	?x→rel.PersonId ?y →rel.ProceedingId
4	?x per_name ?y	?y →persons.Name
5	?x pub_title ?y	?y →papers.Title
6	?x Write ?y	?x→rel_person_paper.PerID ?y →rel_person_paper.PaperID
7	?x per_name ?n ?y pub_title ?t ?x Write ?y	?n→/papers/paper/author/name ?t→/papers/paper/title

The MCUs that will be created are shown in table1. We notice that the last MCU in Table 1 contain all the three triple patterns in its source description. This is because variable ?x and ?y are not return variables, according to the two conditions we defined, they must be in a same MCU.

B. Joining MCUs

In this phase we consider combinations of MCUs, and for each valid combination we create a conjunctive rewriting of the query. The final result is a union of conjunctive queries. Given a set of MCUs, the actual rewriting is constructed as shown in Figure 6.

Continuing the above example, the algorithm will find eight possible rewritings: {1, 2, 3}, {1, 2, 6}, {1, 5, 3}, {1, 5, 6}, {4, 2, 3}, {4, 2, 6}, {4, 5, 3}, {4, 5, 6} and {7}. Take the rewriting {1, 2, 3} for example, which can be translated to SQL query like:

```
SELECT proceeding.Title
FROM person, proceeding, rel
WHERE person.Name = 'Andy'
AND person.PersonId = rel.PersonId
AND proceeding.ProceedingId =
rel.ProceedingId
```

And the rewriting {7} can be translated to XQuery like:

```
for $x in doc("example.xml")/papers
where $x/author/name = 'Andy'
return $x/title
```

Executing each conjunctive query on data source and union the result, we get the result of the example SPARQL query.

```
joinMCUs(Q, M, A)
/* Q is a SPARQL query, M is a set of MCUs, M = {m_1, …, m_n}, where m_i = (Y_i, μ_i'), A is a set of rewritings*/
Initialize A = Φ;
for each minimal subset {m_1,…, m_k} of M such that
    Y_1 ∪ Y_2 ∪ … ∪ Y_k cover all triple patterns of Q.
    Create the conjunctive rewriting Q' contain all relative tables to {m_1,…, m_k}
    Add Q' to A
end for
return A
```

Figure 6. Joining the MCUs

C. Optimizing Rewriting on Relational Queries

We notice that some MCUs have the same triple patterns, which means they have the same semantic. This reminds us that some optimization work can be done. As shown in figure 7, MCU No.1 and No.4 has the same triple patterns, so are the No.2 and No.5, No.3 and No.6. There are eight different ways joining these three kinds of MCUs, as we get in former step. If we union the same kind of MCUs before join them, the number of rewritings can be drastically reduce. In this example, rewritings are reduced from eight to only one.

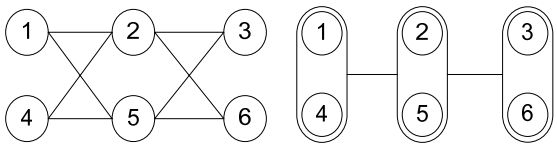


Figure 7. Optimize the rewriting

After the optimization, we get this SQL query:

```
SELECT title
FROM
((SELECT Name as Name, PersonId as
PersonID FROM database1.person) UNION
(SELECT Name as Name, PerID as
PersonID FROM database2.persons)) as
Person,
((SELECT PersonId as PersonID,
ProceedingId as PaperID FROM
database1.rel) UNION (SELECT PerID as
PersonID, PaperID as PaperID FROM
database2.rel_person_Paper)) as Write,
((SELECT Title as Title, ProceedingId
as PaperID FROM database1.proceeding)
UNION (SELECT Title as Title, PaperID
as PaperID FROM database2.papers)) as
Paper
WHERE Person.Name = 'Andy'
AND Person.PersonID = Write.PersonID
AND Paper.PaperID = Write.PaperID
```

VII. Conclusion

The integration of data from multiple heterogeneous sources is an old and well-known research problem for the database and AI research communities. In this paper we considered the problem of integrating heterogeneous data sources using ontology. We discussed the main issues and also solutions. We use ontology as the mediated schema for integration. A novel data integrating architecture based on ontology was presented. We first analyze the similarities and differences among RDF schema, relational model and XML schema, and then discuss how to map relational schema and XML schema to ontology. A data source describing method based on SPARQL is defined. Heterogeneous data source schemas are described using queries defined by SPARQL. Based on RDF Graph model, the semantic of query rewriting is defined and a query rewriting algorithm is presented.

The architecture and approach we provided in this paper can be extended for other data source (i.e. RDF, OWL, web forms, etc.); some of the complex semantic (i.e. CONSTURCT, ASK) are not discussed. We leave these problems for future work.

Acknowledgment

We would like to thank the anonymous reviewers for their many suggestions which helped improve the paper. We would like to acknowledge the support of the National High Technology Research and Development Program of China (No. 2007AA01Z126, 863 Program).

References

[1] A. Y. Halevy, "Answering queries using views: a survey," *In: VLDB Journal*. Vol.10, No.4, pp. 270-294, 2001.

[2] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*. Vol.5, No.2, pp. 199-220, 1993.

[3] J. Wang, Z. Miao, Y. Zhang, and J. Lu, "Semantic integration of relational data using SPARQL," in *International Symposium on Intelligent Information Technology Application*, 2008.

[4] O. Erling and I. Mikhailov, "RDF Support in the Virtuoso DBMS," in *Proceedings of the 1st Conference on Social Semantic Web*, Leipzig, Germany, pp. 59-68, 2007.

[5] C. Bizer and A. Seaborne, "D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs (Poster)," in *3rd International Semantic Web Conference*, Hiroshima, Japan, 2004.

[6] G. Zhou, R. Hull, R. King, and J.-C. Franchitti, "Using object matching and materialization to integrate heterogeneous databases," In *Proceedings. of the 3rd International Conference on Cooperative Information Systems (CoopIS'95)*, Vienna, Austria, pp. 4-18, 1995.

[7] E. Prudhommeaux, "SPASQL: SPARQL Support In MySQL," http://xtech06.usefulinc.com/schedule/paper/156, 2006.

[8] C. P. d. Laborda and S. Conrad, "Bringing Relational Data into the Semantic Web using SPARQL and Relational.OWL," in *Proceedings of the 22nd International Conference on Data Engineering Workshops*, Atlanta, GA, USA, p. 55, 2006.

[9] R. J. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk, "Infosleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments," *SIGMOD Record* vol. 26, pp. 195-206, 1997.

[10] D. Dou and P. LePendu, "Ontology-based Integration for Relational Databases," in *Proceedings of the 2006 ACM symposium on Applied computing*, Dijon, France, pp. 461-466, 2006.

[11] Xquare Fusion. http://xquare.objectweb.org/fusion

[12] Y. Papakonstantinou and V. Vassalos, "The Enosys Markets data integration platform: lessons from the trenches," in *Proceedings of the tenth international conference on Information and knowledge management*, Atlanta, Georgia, USA, pp. 538-540, 2001.

[13] Z. G. Ives, A. Y. Halevy, and D. S. Weld, "An XML query engine for network-bound data," *The VLDB Journal*, vol. 11, pp. 380-402, 2002.

[14] R. Pottinger and A. Halevy, "MiniCon: A Scalable Algorithm for Answering Queries Using Views," *The VLDB Journal*, vol. 10, pp. 182-198, 2001.

[15] J. Hayes, "A graph model for RDF," PhD dissertation, Darmstadt University of Technology, Germany, 2004.

[16] Extensible Markup Language (XML) 1.0 (Fifth Edition), http://www.w3.org/TR/2008/REC-xml-20081126/.
[17] XML Schema, http://www.w3.org/TR/xmlschema-0/.
[18] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying Heterogeneous Information Sources Using Source Descriptions," in *Proceedings of the 22th International Conference on Very Large Data Bases*, San Francisco, CA, USA, pp. 251-262, 1996.
[19] T. Berners-Lee, "Relational Databases on the Semantic Web", http://www.w3.org/DesignIssues/RDB-RDF.html, 1998.
[20] R. R. Swick and H. S. Thompson, "The Cambridge Communiqué." http://www.w3.org/TR/schema-arch, 1999.
[21] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics of SPARQL," Technical Report, Universidad de Chile TR/DCC-2006-17, 2006.

Jinpeng Wang received his BS degree in computer science, from Institute of Command Automation, PLA University of Science and Technology, Nanjing, China in 2005. Now he is a doctoral student in Institute of Command Automation, PLA University of Science and Technology, Nanjing, China major in Computer Science and Technology, with the study direction on Data Integration. At present, he works in related research issues on heterogeneous data integration. Currently, he is focusing on problems of semantic mapping and query rewriting.

Yafei Zhang received the PhD degree in computational linguistics from Fudan University, Shanghai, China in 1992. His research interests are in natural language processing and command automation.

He has written several books and papers about natural language processing and intelligent computing. Now he is a Professor and PhD Supervisor of PLA University of Science and Technology, Nanjing, China. His current research interests are intelligent computing and data integration.

Jianjiang Lu received the PhD degree in computer science from Institute of Command Automation, PLA University of Science and Technology, Nanjing, China in 2002.

He has written several books and papers about fuzzy logics and semantic web. Now he is an Associate Professor of PLA University of Science and Technology, Nanjing, China. His current research interests are intelligent computing and data engineering.

Zhuang Miao received his BS and MS degree in semantic web field from Institute of Communications Engineering, PLA University of Science and Technology, Nanjing, China and the PhD degree in Institute of Command Automation, PLA University of Science and Technology, Nanjing, China in 1999, 2003 and 2007 respectively. His major field of study is semantic web.

He has written several papers about semantic web and intelligent computing. Now he is a lecturer of Institute of Command Automation, PLA University of Science and Technology, Nanjing, China. His current research interests are semantic web and data integration.

Bo Zhou received his BS degree in mathematics, from Nanjing University, Nanjing, China in 2004. Now he is a doctoral student in Institute of Command Automation, PLA University of Science and Technology, Nanjing, China major in Computer Science and Technology, with the study direction on intelligent computing.