Query Translation for Distributed Heterogeneous Structured and Semi-structured Databases

Fahad M. Al-Wasil, N.J. Fiddian, and W.A. Gray

School of Computer Science
Cardiff University
Queen's Buildings
5 The Parade, Roath
Cardiff CF24 3AA
Wales, UK
{Wasil, N.J.Fiddian, W.A.Gray}@cs.cardiff.ac.uk

Abstract. The main purpose of building data integration systems is to facilitate access to a multitude of data sources. A data integration system must contain a module that uses source descriptions in order to reformulate user queries which are posed in terms of the composite global schema, into sub-queries that refer directly to the schemas of the component data sources. In this paper we propose a method for this user query translation task to target distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents (XML documents which have no referenced DTD or XML schema) produced by Internet applications or human-coded. These XML documents can be XML files on local hard drives or remote documents on Web servers. Our method is based on mappings between the master (composite) view and the participating data source schema structures that are defined in a generated XML Metadata Knowledge Base (XMKB).

1 Introduction

Users and application programs in a wide variety of businesses today are increasingly requiring the integration of multiple distributed autonomous heterogeneous data sources [1, 2]. The continuing growth and widespread popularity of the Internet mean that the collection of useful data sources available for public access is rapidly increasing both in number and size. Furthermore, the value of these data sources would in many cases be greatly enhanced if the data they contain could be combined, "queried" in a uniform manner (i.e. using a single query language and interface), and subsequently returned in a machine-readable form. For the foreseeable future, much data will continue to be stored in relational database systems because of the reliability, scalability, tools and performance associated with these systems [3, 4]. However, due to the impact of the Web, there is an explosion in complementary data availability: this data can be automatically generated by Web-based applications and Web services or can be human-coded [5]. Such data is called semi-structured data (ssd) due to its varying degree of structure. In the domain of semi-structured data, the eXtensible Markup Language (XML) is a major data representation as well as data exchange format. XML is a W3C specification [6] that allows creation and

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 73 – 85, 2006.

transformation of a semi-structured document conforming to the XML syntax rules and having no referenced DTD or XML schema. Such a document has metadata buried inside the data and is called a well-formed XML document. The metadata content of XML documents enables automated processing, generation, transformation and consumption of semi-structured information by applications. Much interesting and useful data can be published as a well-formed XML document by Web-based applications and Web services or by human-coding.

Hence, building a data integration system that provides unified access to semantically and structurally diverse data sources is highly desirable to link structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by Internet applications or human-coded [7, 8]. These XML documents can be XML files on local hard drives or remote documents on Web servers. The data integration system has to find structural transformations and semantic mappings that result in correct merging of the data and allow users to query the so-called mediated schema [9]. This linking is a challenging problem since the pre-existing databases concerned are typically autonomous and located on heterogeneous hardware and software platforms. In this context, it is necessary to resolve several conflicts caused by the heterogeneity of the data sources with respect to data model, schema or schema concepts. Consequently, mappings between entities from different sources representing the same real-world objects have to be defined. The main difficulty is that the related data from different sources may be represented in different formats and in incompatible ways. For instance, the bibliographical databases of different publishers may use different formats for authors' or editors' names (e.g. full name or separated first and last names), or different units for prices (e.g. dollars, pounds or euros). Moreover, the same expression may have a different meaning, or the same meaning may be specified by different expressions. This implies that syntactical data and metadata alone cannot provide sufficient semantics for all potential integration purposes. As a result, the data integration process is often very labour-intensive and demands more computing expertise than most application users have. Therefore, semi-automated approaches seem the most promising way forward, where mediation engineers are given an easy tool to describe mappings between the integrated (integrated and master are used interchangeably in this paper) view and local schemas, to produce a uniform view over all the participating local data sources [10].

XML is becoming the standard format to exchange information over the Internet. The advantages of XML as an exchange model - such as rich expressiveness, clear notation and extensibility - make it an excellent candidate to be a data model for the integrated schema. As the importance of XML has increased, a series of standards has grown up around it, many of which were defined by the World Wide Web Consortium (W3C). For example, the XML Schema language provides a notation for defining new types of XML elements and XML documents. XML with its self-describing hierarchical structure and the language XML Schema provide the flexibility and expressive power needed to accommodate distributed and heterogeneous data. At the conceptual level, they can be visualized as trees or hierarchical graphs.

In [11] we proposed and described a System to Integrate Structured and Semi-structured Databases (SISSD) through a mediation layer. Such a layer is intended to

combine and query distributed heterogeneous structured data residing in relational databases with semi-structured data held in well-formed XML documents (that conform to the XML syntax rules but have no referenced DTD or XML schema) produced by Internet applications. We investigated how to establish and evolve an XML Metadata Knowledge Base (XMKB) incrementally to assist the Query Processor in mediating between user queries posed over the master view and the underlying distributed heterogeneous data sources. The XMKB is built in a bottom-up fashion by extracting and merging incrementally the metadata of the data sources. It contains and maintains data source information (names, types and locations), meta-information about relationships of paths among data sources, and function names for handling semantic and structural discrepancies. The associated SISSD system automatically creates a GUI tool for meta-users (who do the metadata integration) to describe mappings between the master view and local data sources by assigning index numbers and specifying conversion function names. From these mappings the SISSD produces the corresponding XML Metadata Knowledge Base (XMKB), which is capable of supporting the generation of queries to local data sources from user queries posed over the master view. The GUI tool parses the master view to generate an index number for each element and parses local schema structures to generate a path for each element. Mappings assign indices to match local elements with corresponding master elements and to names of conversion functions, which can be built-in or user-defined functions. The XMKB is derived based on the mappings by combination over index numbers.

We have proposed a generic mechanism to compute index numbers for the master view elements. By applying this mechanism, a unique index number is generated for each element in an XML document whatever the nesting complexity of the document. We have also described several mapping cases between master view and local schema structure elements (e.g. One-to-One, One-to-Many and Many-to-One) and how to resolve structural and semantic conflicts that may occur between elements.

This system is flexible: users can assemble any virtual master view they want from the same set of data sources depending on their interest. It also preserves local autonomy of the local data sources, thus these data sources can be handled without rebuilding or modification. The SISSD uses the local-as-view approach to map between the master view and the local schema structures. This approach is well-suited to supporting a dynamic environment, where data sources can be added to or removed from the system without the need to restructure the master view. The XML Metadata Knowledge Base (XMKB) is evolved and modified incrementally when data sources are added to or removed from the system, without the need to regenerate it from scratch.

This paper concentrates on the problem of querying a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents. The important aspect is to develop a technique to seamlessly translate user queries over the master view into sub-queries - called local queries - fitting each participating data source, by exploiting the mapping information stored in the XMKB.

User queries are formulated in XQuery (a powerful universal query language for XML) FLWR (short for For-Let-Where-Return) expressions and processed

according to the XMKB, by generating an executable (sub-) query for each relevant local data source.

The rest of this paper is organized as follows. The next section presents related work. The SISSD architecture and the internal architecture of its Query Processor (QP) are described in section 3. Section 4 presents the structure, content and organization of knowledge in the XMKB. Section 5 summarises the query translation process in algorithmic form. Finally, we present conclusions in section 6.

2 Related Work

Data integration has received significant attention since the early days of databases. In recent years, there have been several projects focusing on heterogeneous information integration. Most of them are based on a common mediator architecture [12] such as Garlic [13], the Information Manifold [14], Disco [15], Tsimmis [16], Yat [17], Mix [18], MedMaker [19] and Agora [20]. The goal of such systems is to provide a uniform user interface to query integrated views over heterogeneous data sources. A user query is formulated in terms of the integrated view; to execute the query, the system translates it into sub-queries expressed in terms of the local schemas, sends the sub-queries to the local data sources, retrieves the results, and combines them into the final result provided to the user. Data integration systems can be classified according to the way the schemas of the local data sources are related to the global, unified schema. A first approach is to define the global schema as a view over the local schemas: such an approach is called global-as-view (GAV). The opposite approach, known as local-as-view (LAV), consists of defining the local sources as views over the global schema [21].

Now consider query processing. In the GAV approach, translating a query over the global schema into queries against the local schemas is a simple process of view unfolding. In the case of LAV, the query over the global schema needs to be reformulated in terms of the local data source schemas; this process is traditionally known as "rewriting queries using views" and is a known hard problem [22].

Projects like Garlic, Disco, Tsimmis, Mix, MedMaker and Yat all adopt the GAV approach, and therefore do not compare directly to our system since we use the LAV approach. Projects like the Information Manifold and Agora are integration systems with a LAV architecture; however, in the Information Manifold the local and global schemas are relational, while the Agora system supports querying and integrating data sources of diverse formats, including XML and relational sources under an XML global schema, but assumes explicit schemas for XML data sources.

SilkRoute [23] and XPERANTO [4, 24] focus on exporting relational databases under an XML interface. Since the mapping is done from tuples to XML, these projects adopt the GAV approach; also, they can only integrate relational data sources. By contrast, our integration approach can handle diverse data sources (XML and relational), not just relational. Also SISSD follows the Information Manifold and Agora systems by adopting the LAV approach.

The LAV approach provides a more flexible environment to meet users' evolving and changing information requirements across the disparate data sources available over the global information infrastructure (Internet). It is better suited and scalable for

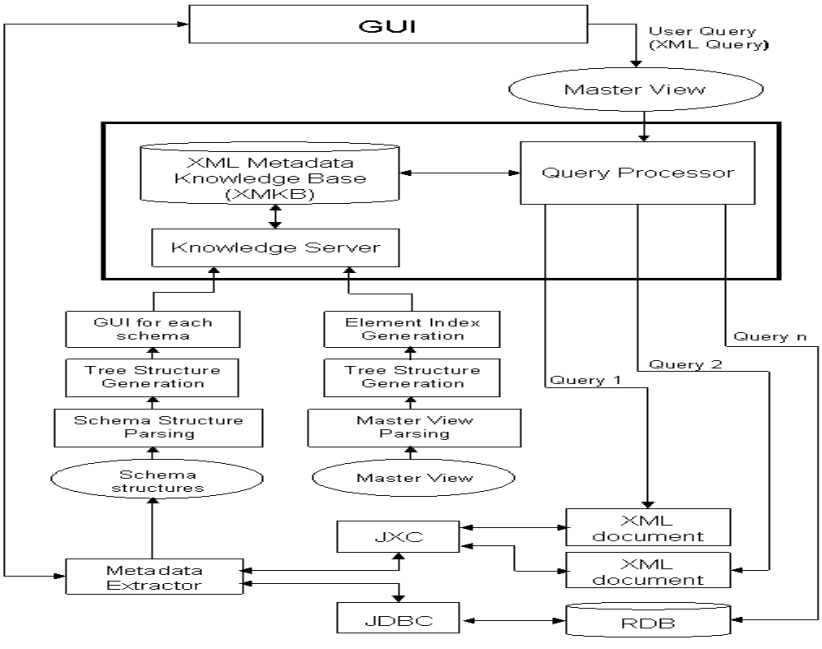


Fig. 1. The SISSD Architecture

integrating a large number of autonomous read-only data sources accessible over communication networks. Furthermore the LAV approach provides a flexible environment able to accommodate the continual change and update of data source schemas, especially suitable for XML documents on Web servers since these remote documents are not static and are often subject to frequent update.

3 The SISSD Architecture and Components

In this section, we present an overview of the SISSD architecture and summarize the functions of the main components. The architecture we adopt is depicted in Figure 1. Its main components are the Metadata Extractor (MDE), the Knowledge Server (KS) and the Query Processor (QP).

3.1 Metadata Extractor (MDE)

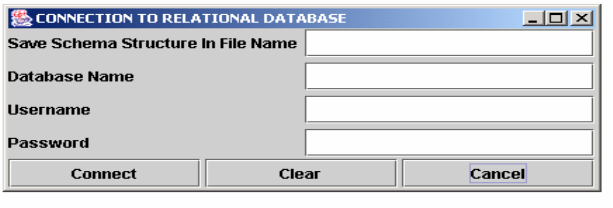
The MDE needs to deal with heterogeneity at the hardware, software and data model levels without violating the local autonomy of the data sources. It interacts with the data sources via JDBC (Java Database Connectivity) if the data source is a relational database or via JXC (Java XML Connectivity) if the data source is an XML document. The MDE extracts the metadata of all data sources and builds a schema structure in XML form for each data source.

We developed JXC using a JDOM (Java Document Object Model) interface to detect and extract the schema structure of a well-formed XML document (that conforms to the XML syntax rules but has no referenced DTD or XML schema), where the metadata are buried inside the data.

3.1.1 Schema Structures

Typically, the heterogeneous data sources use different data models to store their data (e.g. relational model and XML model). This type of heterogeneity is referred to as syntactic heterogeneity. The solution commonly adopted to overcome syntactic heterogeneity is to use a common data model and to map all schemas to this common model. The advantages of XML as an exchange model make it a good candidate to be the common data model and for supporting the integrated data model. The metadata extracts generated on top of the data sources by using this data model are referred to as schema structures. We define a simple XML Data Source Definition Language (XDSDL) for describing and defining the relevant identifying information and the data structure of a data source. The XDSDL is represented in XML and is composed of two parts. The first part provides a description of the data source name, location and type (relational database or XML document). The second part provides a definition and description of the data source structure and content. The emphasis is on making these descriptions readable by automated processors such as parsers and other XML-based tools. This language can be used for describing the structure and content of relational databases and well-formed XML documents which have no referenced DTD or XML schema.

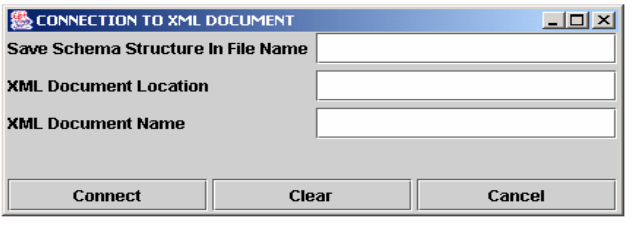
For relational databases the MDE employs JDBC to access the DB without making any changes to it. The MDE accepts the information necessary to establish a connection to a DB to retrieve the metadata of its schema and uses the XDSDL to build the target schema structure for that DB, together with necessary information such as the DB location (URL), where to save the schema structure, the User ID and Password.



It opens a connection to that DB through a JDBC driver. Opening this connection enables SQL queries to be issued to and results to be retrieved from the DB. Once the connection is established, the MDE retrieves the names of all the tables defined in the accessed DB schema and then uses the XDSDL to define these tables as elements in the target schema structure. Furthermore, for each table the MDE extracts and analyses the attribute names, then defines these attributes as child elements for that table element in the target schema structure using the XDSDL.

For XML documents the MDE employs JXC to access the document without making any changes to it. The MDE accepts the information necessary to establish a connection to a well-formed XML document to retrieve the metadata of its schema

where the metadata are buried inside the data. It then uses the XDSDL to build the target schema structure for that XML document, together with necessary information such as the document location (URL), where to save the schema structure, and the document name.



It opens a connection to that XML document through a JDOM interface. Once the connection is established, the JXC automatically tracks the structure of the XML document, viz. each element found in the document, which elements are child elements and the order of child elements. The JXC reads the XML document and detects the start tag for the elements. For each start tag, the JXC checks if this element has child elements or not: if it has then this element is defined as a complex element in the target schema structure using the XDSDL, otherwise it is defined as a simple element by the MDE. The defined elements in the target schema structure take the same name as the start tags.

3.2 Knowledge Server (KS)

The Knowledge Server (KS) is the central component of the SISSD. Its function is to establish, evolve and maintain the XML Metadata Knowledge Base (XMKB), which holds information about the data sources and provides the necessary functionality for its role in assisting the Query Processor (QP). The KS creates a GUI tool for meta-users to do metadata integration by building the XML Metadata Knowledge Base (XMKB) that comprises information about data structures and semantics. This information can then be used by the Query Processor (QP) to automatically rewrite a user query over the master view into sub-queries called local queries, fitting each local data source, and to integrate the results.

3.3 Query Processor (QP)

The Query Processor (QP) is responsible for receiving a user query (master query) over a master view to process it and return the query result to the user. The master view provides the user with the elements on which the query can be based. The QP gives flexibility to the user to choose the master view that he/she wants to pose his/her query over and then automatically selects the appropriate XMKB that will be used to process any query posed over this master view. The query language that our QP supports is XQuery FLWR expressions. XQuery is the standard XML query language being developed by the W3C [25]. It is derived from Quilt, an earlier XML query language designed by Jonathan Robie together with IBM's Don Chamberlin, co-inventor of SQL, and Daniela Florescu, a well-known database researcher. XQuery is

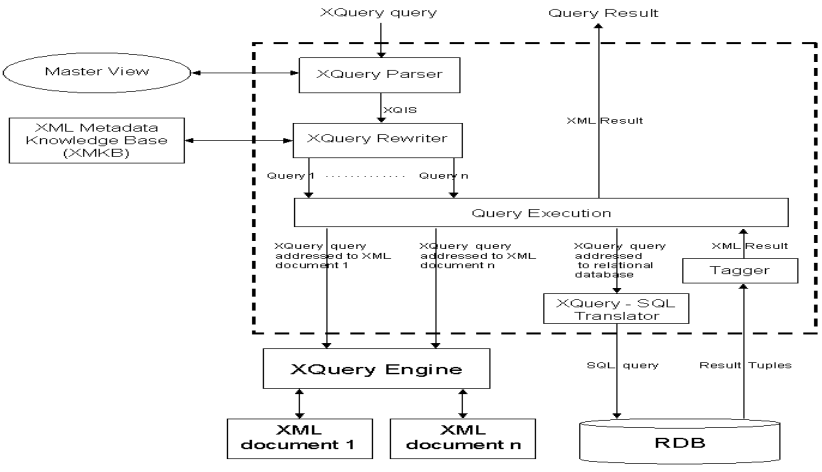


Fig. 2. Internal Architecture of the Query Processor

designed to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. It can be used to query XML data that has no schema at all, or that is governed by a W3C standard XML Schema or by a Document Type Definition (DTD).

XQuery is centered on the notion of expression; starting from constants and variables, expressions can be nested and combined using arithmetic, logical and list operators, navigation primitives, function calls, higher order operators like sort, conditional expressions, element constructors, etc. For navigating in a document, XQuery uses path expressions, whose syntax is borrowed from the abbreviated syntax of XPath. The evaluation of a path expression on an XML document returns a list of information items, whose order is dictated by the order of elements within the document (also called document order).

A powerful feature of XQuery is the use of FLWR expressions (For-Let-Where-Return). The *for-let* clause makes variables iterate over the result of an expression or binds variables to arbitrary expressions, the *where* clause allows specification of restrictions on the variables, and the *return* clause can construct new XML elements as output of the query. In general, an XQuery query consists of an optional list of namespace definitions, followed by a list of function definitions, followed by a single query expression.

Supporting FLWR expressions for querying a master view makes it easy to translate the sub-queries directed at relational databases into SQL queries since syntactically, FLWR expressions look similar to SQL select statements and have similar capabilities, only they use path expressions instead of table and column names.

The internal architecture of the Query Processor (QP) is shown in Figure 2. It consists of five components: XQuery Parser, XQuery Rewriter, Query Execution, XQuery-SQL Translator, and Tagger. The core of the QP and the primary focus of

this paper is the XQuery Rewriter. This component rewrites the user query posed over the master view into sub-queries which fit each local data source, by using the mapping information stored in the XMKB. The main role played by each of the components in Figure 2 is described below.

- **XQuery Parser:** parses a given XQuery FLWR expression in order to check it for syntactic correctness and ensure that the query is valid and conforms to the relevant master view. Also the parser analyses the query to generate an XQuery Internal Structure (XQIS) which contains the XML paths, variables, conditions and tags present in the query, then passes it to the XQuery Rewriter.
- **XQuery Rewriter:** Takes the XQIS representation of a query, consults the XMKB to obtain the local paths corresponding to the master paths and function names for handling semantic and structural discrepancies, then produces semantically equivalent XQuery queries to fit each local data source. That is, wherever there is a correspondence between the paths in the master view and local schema structures concerned (otherwise the local data source is ignored).
- **Query Execution:** Receives the rewritten XQuery queries, consults the XMKB to determine each data source's location and type (relational database or XML document), then sends each local query to its corresponding query engine, to execute the query and return the results.
- **XQuery-SQL Translator:** Translates an XQuery query addressed to a relational database into the SQL query needed to locate the result, then hands the query over to the relational database engine to execute it and return the result in tabular format through the Tagger.
- **Tagger:** Adds the appropriate XML tags to the tabular SQL query result to produce structured XML documents for return to the user.

4 The Structure of the XMKB

The XML Metadata Knowledge Base (XMKB) is an XML document composed of two parts. The first part contains information about data source names, types and locations. The second part contains meta-information about relationships of paths among data sources, and function names for handling semantic and structural discrepancies. The XMKB structure is shown in Figure 3. The <*DS_information*> element here contains data source names, types and locations. The <*DS_information*> element has one attribute called number which holds the number of data sources participating in the integration system (3 in the example shown). Also the <*DS_information*> element has child elements called <*DS_Location*>. Each <*DS_Location*> element contains the data source name, its type (relational database or XML document) as an attribute value and the location of the data source as an element value. This information is used by the Query Processor to specify the type of generated sub-query (SQL if the data source type is relational database, or XQuery if it is XML document) and the data source location that the system should submit the generated sub-query to.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <XMKB>
  - <DS_information number="3">
      <DS_Location name="books.xml" type="XML document">http://www.w3schools.com/xquery</DS_Location>
      <DS_Location name="bib.xml" type="XML document">C:\prototype\doc</DS_Location>
      <DS_Location name="SCMFMA" type="Relational Database">jdbc:oracle:thin:@helot:1521:oracle9</DS_Location>
    </DS_information>
  - <Med_component>
    - <source path="/book">
        <target name="books.xml" fun="Null">/bookstore/book</target>
        <target name="bib.xml" fun="Null">/bib/book</target>
        <target name="SCMFMA" fun="Null">/scmfma/book</target>
      </source>
    - <source path="/book/price">
        <target name="books.xml" fun="RateExchange">/bookstore/book/price</target>
        <target name="bib.xml" fun="RateExchange">/bib/book/price</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">/bib/book/author</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author/full_name">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">Null</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author/full_name/first_name">
        <target name="books.xml" fun="firstName">/bookstore/book/author</target>
        <target name="bib.xml" fun="Null">/bib/book/author/first</target>
        <target name="SCMFMA" fun="firstName">/scmfma/book/author</target>
      </source>
```

Fig. 3. A sample XMKB document

The <*Med_component*> element in Figure 3 contains the path mappings between the master view elements and the local data source elements, and the function names for handling semantic and structural discrepancies. The master view element paths are called <*source*> elements, while corresponding element paths in local data sources are called <*target*> elements. The <*source*> elements in the XMKB document have one attribute called path which contains the path of the master view elements. These <*source*> elements have child elements called <*target*> which contain the corresponding paths for the master view element paths in each local data source, or null if there is no corresponding path. The <*target*> elements in the XMKB document have two attributes. The first one is called *name* and contains the name of the local data source, while the second is called *fun* and contains the function name that is needed to resolve semantic and structural discrepancies between the master view element and the local data source element concerned, or null if there is no discrepancy.

5 The Query Translation Process

From the foregoing descriptions of the SISSD Query Processor (QP) component architecture (section 3.3) and the XML Metadata Knowledge Base (XMKB) organization and contents (section 4), we are now in a position to summarise the query translation (rewriting) process carried out at the heart of our system by the QP module. We do so in algorithmic form as follows, c.f. Figure 2 earlier. The algorithm is both conceptually simple and generally applicable. We have successfully implemented and tested it on a variety of relational and XML data source integration examples in our prototype SISSD system.

Algorithm. Master query translation process

Input: Master View, Master XQuery query *q*, and XMKB

Output: local sub-queries *q1*, *q2*…, *qn*

Step1: **parse** *q;*

Step2: **get** global paths *g1*, *g2*…., *gn* from Master View;

Step3: **read** XMKB;

Step4: **identify** the number of local data sources participating in the integration system, their locations and types;

Step5: **for each** data source *S*i **do**
for each global path *ge* in *q* **do**
if the corresponding local path *le* not null **then**
get *le*;
if the function name *fe* not null **then**
get *fe*;
end if
else
no query generated for this local data source *S*i ;
end if
end for
replace *g1* by *l1* with *f1, g2* by *l2* with *f2 ..., gn* by *ln* with *fn,* in *qi*;
if data source type is relational database **then**
convert *qi* XQuery into SQL;
end for

Step6: **execute** the generated local query *qi* by sending it to the corresponding local data source engine, and return the result, with XML tags added to SQL tables.

6 Conclusions

In this paper, we have described an approach for querying a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents (XML documents which have no referenced DTD or XML schema) produced by Internet applications or human-coded. These XML documents can be XML files on local hard drives or remote documents on Web servers. Our method is based on mappings between the master view and the participating data source schema structures that are defined in a generated XML Metadata Knowledge Base (XMKB). The basic idea is that a query posed to the integrated system, called a master query, is automatically rewritten into sub-queries called local queries which fit each local data source, using the information stored in the XMKB. This task is accomplished by the Query Processor module. Such an approach produces a system capable of querying across a set of heterogeneous distributed structured and semi-structured data sources. We have developed a

prototype system to demonstrate that the ideas explored in the paper are sound and practical, also clearly convenient from a user standpoint.

As a result, our system can easily incorporate a large number of relational databases and XML data sources from the same domain. However, most of the existing data integration systems concerned with XML documents are interested in documents that use DTD (Document Type Definition) or XML Schema language for describing the schemas of the participating heterogeneous XML data sources. We have investigated and used XML documents which have no referenced DTD or XML schema, rather the schema metadata are buried inside the document data. This paper has shown that querying a set of distributed heterogeneous structured and semi-structured data sources of this form and in this way is possible; its relevance in the Internet/Web context is readily apparent.

In addition to this, our Query Processor (QP) has been implemented using Java, JDOM API, and the JavaCC compiler. It accepts FLWR expressions as an XML query language, this is a subset of XQuery which supports the basic requirements of our approach, particularly the uniform querying of heterogeneous distributed structured (relational database) and semi-structured (well-formed XML document) data sources.

References

1. Hu, G. and H. Fernandes, Integration and querying of distributed databases, in Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2003), October 27-29, 2003: Las Vegas, NV, USA. p. 167-174.
2. Segev, A. and A. Chatterjee, *Data manipulation in heterogeneous databases.* Sigmod Record, December 1991. **20(4)**: p. 64-68.
3. Funderburk, J.E., et al., *XTABLES: Bridging Relational Technology and XML.* IBM Systems Journal, 2002. **41(4)**: p. 616-641.
4. Shanmugasundaram, J., et al., Efficiently Publishing Relational Data as XML Documents, in Proceedings of the 26th International Conference on Very Large Databases (VLDB2000), September 2000: Cairo, Egypt. p. 65-76.
5. Lehti, P. and P. Fankhauser, XML data integration with OWL: Experiences & challenges, in Proceedings of the International Symposium on Applications and the Internet (SAINT 2004), 2004: Tokyo, Japan. p. 160-170.
6. World Wide Web Consortium, *http://www.w3.org/TR/2004/REC-xml-20040204/.* Extensible Markup Language (XML) 1.0 W3C Recommendation, third edition, February 2004.
7. Gardarin, G., F. Sha, and T. Dang-Ngoc, XML-based Components for Federating Multiple Heterogeneous Data Sources, in ER '99: Proceedings of the 18th International Conference on Conceptual Modeling, 1999, Springer-Verlag. p. 506-519.
8. Lee, K., J. Min, and K. Park, A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources, in HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02) - Volume 7. 2002, IEEE Computer Society. p. 202-210.
9. Kurgan, L., W. Swiercz, and K. Cios, Semantic Mapping of XML Tags using Inductive Machine Learning, in Proceedings of the International Conference on Machine Learning and Applications - ICMLA '02. 2002: Las Vegas, Nevada, USA.

10. Young-Kwang, N., G. Joseph, and W. Guilian, A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases, in Proceedings of the Confederated International Conferences DOA, CoopIS and ODBASE. October 2002, LNCS 2519, Springer, p. 1332-1344.: Irvine CA.
11. Al-Wasil, F.M., W.A. Gray, and N.J. Fiddian, Establishing an XML Metadata Knowledge Base to Assist Integration of Structured and Semi-structured Databases, in ADC '2006: Proceedings of The Seventeenth Australasian Database Conference. January 16th - 19th 2006: Tasmania, Australia.
12. Wiederhold, G., *Mediators in the Architecture of Future Information System.* IEEE Computer, March 1992. **25(3)**: p. 38-49.
13. Carey, M.J., et al., *Towards heterogeneous multimedia information systems: the Garlic approach*, in *RIDE '95: Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95).* 1995, IEEE Computer Society. p. 124-131.
14. Kirk, T., et al., The Information Manifold, in Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, p. 85-91. March, 1995.: Stanford University, Stanford, CA.
15. Tomasic, A., L. Raschid, and P. Valduriez, *Scaling access to heterogeneous data sources with DISCO.* IEEE Transactions on Knowledge and Data Engineering, 1998. **10**(5): p. 808-823.
16. Ullman, J.D., Information Integration Using Logical Views, in ICDT '97: Proceedings of the 6th International Conference on Database Theory. 1997, Springer-Verlag. p. 19-40.
17. Christophides, V., S. Cluet, and J. Simèon, On wrapping query languages and efficient XML integration, in Proceedings of ACM SIGMOD Conference on Management of Data. May 2000.: Dallas, Texas, USA.
18. Baru, C., et al., XML-based information mediation with MIX, in SIGMOD '99: Proceedings of ACM SIGMOD International Conference on Management of Data. 1999, ACM Press. p. 597-599.
19. Papakonstantinou, Y., H. Garcia-Molina, and J.D. Ullman, MedMaker: A Mediation System Based on Declarative Specifications, in ICDE '96: Proceedings of the 12th International Conference on Data Engineering. 1996, IEEE Computer Society. p. 132-141.
20. Manolescu, I., D. Florescu, and D. Kossmann, Answering XML Queries over Heterogeneous Data Sources, in Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). September 2001: Rome, Italy.
21. Lenzerini, M., Data integration: a theoretical perspective, in Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2002: Madison, Wisconsin.
22. Levy, A., et al., Answering queries using views, in Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. 1995: San Jose, CA, USA.
23. Fernndez, M., W.-C. Tan, and D. Suciu, SilkRoute: Trading between Relations and XML, in Proceedings of the Ninth International World Wide Web Conference. May 15 - 19 2000: Amsterdam.
24. Shanmugasundaram, J., et al., Querying XML Views of Relational Data, in proceedings of the 27th International Conference on Very Large Data Bases (VLDB). September 2001: Rome, Italy.
25. World Wide Web Consortium, *http://www.w3.org/TR/xquery/.* XQuery 1.0: An XML Query Language, W3C Working Draft, November 2003.