A MEDIATION LAYER FOR HETEROGENEOUS XML SCHEMAS

Abdelsalam Almarimi[1], Jaroslav Pokorny[2]

Abstract

This paper describes an approach for mediation of heterogeneous XML schemas. Such an approach is proposed as a tool for XML data integration system. A global XML schema is specified by the designer to provide a homogeneous view over heterogeneous XML data. An XML mediation layer is introduced to manage: (1) establishing appropriate mappings between the global schema and the schemas of the sources; (2) querying XML data sources in terms of the global schema. The XML data sources are described by XML Schema language. The former task is performed through a semi-automatic process that generates local and global paths. A tree structure for each XML schema is constructed and represented by a simple form. This is in turn used for assigning indices manually to match local paths to corresponding global paths. By gathering all paths with the same indices, the equivalent local and global paths are grouped automatically, and an XML Metadata Document is constructed. An XML Query Translator for the latter task is described to translate a global user query into local queries by using the mappings that are defined in the XML Metadata Document.

1. Introduction

XML [11] 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 the best candidate for supporting the integrated data model. Tools and infrastructures for data integration are required due to the increasing number of distributed heterogeneous data sources on-line.

However, modern business often needs to combine heterogeneous data from different data sources. Therefore, tools are needed to mediate between user queries and heterogeneous data sources to

[1] Czech Technical University, FEL, Department of Computers, Karlovo nam 13, Praha 3, Czech Republic; (email: belgasem@cslab.felk.cvut.cz)
[2] Charles University, MFF, Department of Software Engineering, Malostranske nam. 25, Praha 1, Czech Republic; (email: pokorny@ksi.ms.mff.cuni.cz)

translate such queries into local queries. 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, XML Schema language [13,14,15] 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 distributive and heterogeneous data. At the conceptual level, they can be visualized as trees or hierarchical graphs.

Regardless the used schema description language, each schema integration process involves three main stages: conflict analysis, conflict resolution, and schema merging. During conflict analysis, differences in the schemas are identified. In the second stage the conflicts are resolved. Finally, the schemas are merged into a single global schema using the decisions made during the previous stage. 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. Therefore, the mapping between entities from different sources representing the same real-world objects has to be defined. The main difficulty is that the data at different sources may be represented in different formats and in incompatible ways. For example, the bibliographical databases of different publishers may use different formats of authors' or editors' names or different units of prices. Moreover, the same expression may have a different meaning, and the same meaning may be specified by different expressions. To integrate or reconcile schemas we must understand how they correspond. If the schemas are to be integrated, their corresponding information should be reconciled and modeled in one consistent way.

This paper mainly refers to the problem of integrating heterogeneous XML data sources. We propose a method to combine and query XML documents through a mediation layer. Such a layer is proposed to describe the mappings between global XML schema and local heterogeneous XML schemas. It produces a uniform interface over the local XML data sources and provides the required functionality to query these sources in a uniform way. It involves two important units: the XML Metadata Document (XMD) and the Query Translator. The XMD is an XML document containing metadata, in which the mappings between global and local schemas are defined. The XML Query Translator which is an integral part of the system is introduced to translate a global user query into local queries by using the mappings that are defined in the XMD.

The rest of the paper is organized as follows. The next section presents the related work. Section 3 introduces the architecture of the XML data integration system. In section 4 we present XML schema processing. The mediation process of XSDs is introduced in section 5. Section 6 describes the query translator unit. Finally, we conclude the paper.

2. Related work

Data integration has received significant attention since the early days of databases. In the recent years, there have been several works focusing on heterogeneous information integration. Most of them are based on common mediator architecture [6]. In this architecture, mediators provide a uniform user interface to views of heterogeneous data sources. They resolve queries over global concepts into subqueries over data sources. Mainly, they can be classified into structural approaches and semantic approaches.

In *structural approaches*, local data sources are assumed as crucial. The integration is done by providing or automatically generating a global unified schema that characterizes the underlying data sources. On the other hand, in *semantic approaches*, integration is obtained by sharing a common

ontology among the data sources. According to the mapping direction, the approaches are classified into two categories: global-as-view and local-as-view [9]. In *global-as-view* approaches, each item in the global schema is defined as a view over the source schemas. In *local-as-view* approaches, each item in each source schema is defined as a view over the global schema. The local-as-view approach better supports a dynamic environment, where data sources can be added to the data integration system without the need to restructure the global schema.

There are several well-known research projects and prototypes such as Garlic [8], Tsimmis [7], MedMaker [17], and Mix [4] are structural approaches and take a global-as-view approach. A common data model is used, e.g., OEM (Object Exchange Model) in Tsimmis and MedMaker. Mix uses XML as the data model; an XML query language XMAS was developed and used as the view definition language there. DDXMI [16] (for Distributed Database XML Metadata Interface) builds on XML Metadata Interchange. DDXMI is a master file including database information, XML path information (a path for each node starting from the root), and semantic information about XML elements and attributes. A system prototype has been built that generates a tool to do the metadata integration, producing a master DDXMI file, which is then used to generate queries to local databases from master queries. In this approach local sources were designed according to DTD definitions. Therefore, the integration process is started from the DTD parsing that is associated to each source.

Many efforts are being made to develop semantic approaches, based on RDF (Resource Description Framework) and knowledge-based integration [3]. Several ontology languages have been developed for data and knowledge representation to assist data integration from a semantic perspective, such as Ontolingua [1]. F-logic [11] is employed to represent knowledge in the form of a domain map to integrate data sources at the conceptual level. An ontology based approach [5] is one from many other researches which use ontologies to create a global schema

We classify our system as a structural approach and differ from the others by following the local-as-view approach. The XML Schema language is adopted in our work instead of DTD grammar language, which has limited applicability. While only simple cases of heterogeneity conflicts among elements were handled in the paper [2], this work involves more features of XML schema components; we handle more mapping cardinality cases involving attributes in which the core purpose is to provide more information about the elements.

3. System architecture overview

The architecture of the data integration system is presented in Figure 1. The data sources that we are interested in are XML documents satisfying different XML schemas. The main component of the system is the mediation layer, which comprises the XML Metadata Document (XMD) and the Query Translator.

The XMD is an XML document containing metadata, in which the mappings between global and local schemas are defined. The main objective is that when a *global query* over the global XML schema is posed, it is automatically translated by the Query Translator unit to subqueries, called *local queries*, which fit each local source format using the information stored in XMD. A GUI tool is also involved, which is a simple form used to simplify the mapping process among schemas.

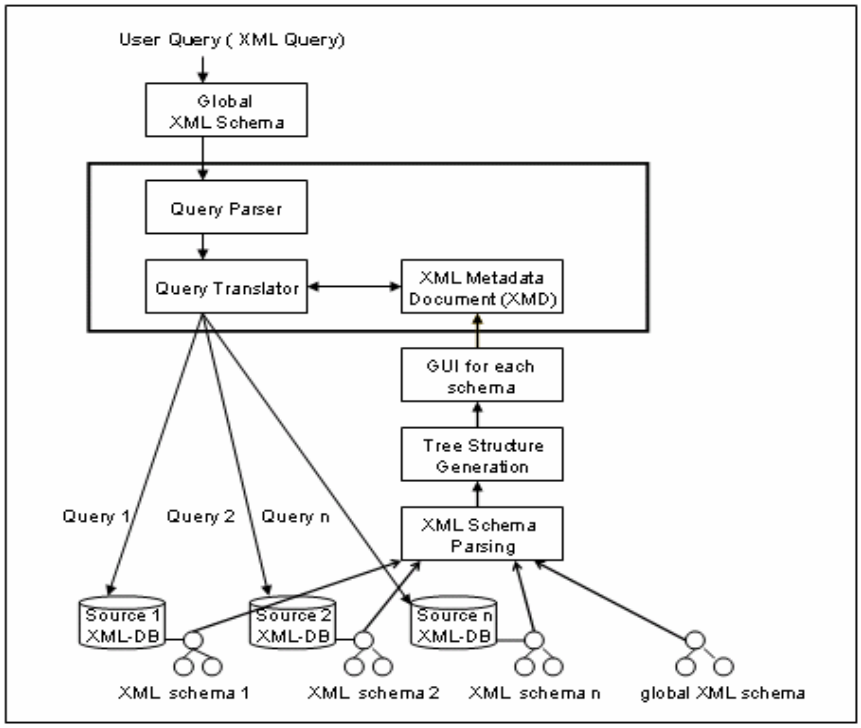

Figure 1. Data Integration System Architecture.

4. XML schema processing

The XML schema is itself an XML document, which we denote as XSD (XML schema document). It is a sequence of components where each component is an attribute, or an element or a simple type or complex type. The JDOM API is used for reading XSDs in memory.

4.1. XSD modeling

We model XSD as a tree structure whose nodes are components of the corresponding local sources. Each component corresponds to the occurrence of a tag, to the occurrence of an attribute, to the content of tag, and so on. Here, we only consider acyclic XML schemas with attributes. To clarify our approach, we introduce an example in which three publishers' database sites are used. Our objective is to create a global view over these heterogeneous sites to be used for query purposes. The publishers are Addison Wesley (AW), Prentice Hall (PH), and Wiley. The structure of each site was studied carefully and their XML schemas were defined. Although AW, PH, and Wiley all contain book information, the data structures are different. Let us assume that the author information of the global schema is divided into first name and last name, while in the local sources it is represented as full name. Also the price unit of the local sources is the dollar, while the global schema uses the euro. In addition, the book format of AW is represented as a single element, while in Wiley it is divided into two elements: CoverType and Pages.

We present in Figure 2 a part of the tree structures of the schemas that are used in the example. In this work, the process of constructing the global schema is not automated. The global XML schema is specified by the designer, and the basic notions in the domain are described.

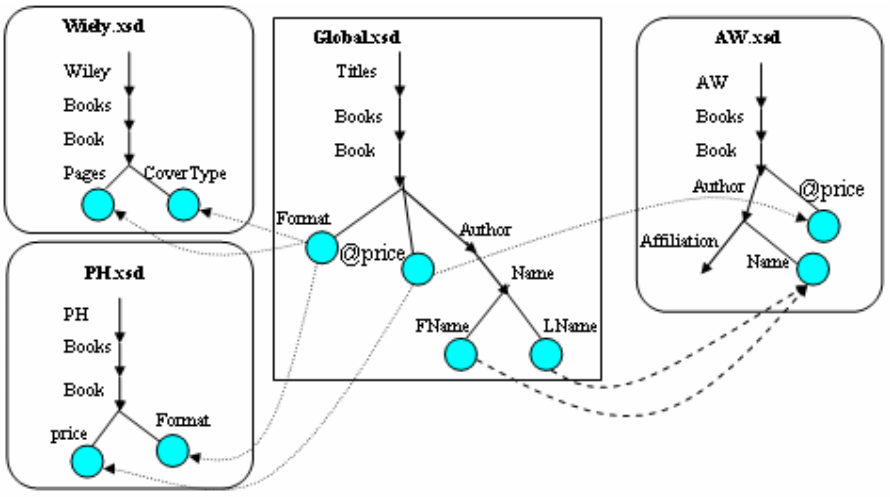

Figure 2. A part of the tree structure for XSDs.

4.2 Extracting XSD components

JDOM is a tree-based, pure Java API for parsing, creating, and manipulating XML documents. It provides a full document view with random access. Once a document has been loaded into memory, whether by creating it from scratch or by parsing it from a stream, it can be easily processed by JDOM. Thus the entire tree of XSD is available at any time. In fact, JDOM itself does not include a parser. Instead it depends on a SAX parser [10], which can be used to parse documents and build JDOM models from them. Once we have parsed an XSD, a JDOM tree model (`a document object`) is formed which contains the entire components of the XSD. Figure 3 shows an example of an `object` tree structure. In this model, each component is represented by its *name, value, and type,* respectively. In turn, we need to search such a structure and extract out the components that we are interested in. Let ELEMENTS and ATTRIBUTES be a set of elements and attributes, respectively, of the `document object`. Formally, we introduce a function:

$$\text{CHILD: COMPONENT} \rightarrow \wp(\text{COMPONENT}),$$

where COMPONENT = ELEMENTS $\cup$ ATTRIBUTES, which assigns a multiset of child components to each component in an XSD[1]. Basically, the CHILD function is founded to materialize the XSD components that are needed. In turn, it can be navigated to generate a unique path for each node starting from the root.

The process of extracting XSD components comprises the following steps:

1. A JDOM tree model is formed for each XSD.
2. For each XSD object, the *value* of each components name (exclude the *name* and *type*) is extracted and a new tree data structure *x* is constructed.
3. A unique number is assigned to each node of *x* to resolve naming conflicts.
4. A depth-first traversal is performed on *x* and the CHILD function is materialized.

[1] In our implementation CHILD is realized by a JAVA 2 hash table assigning to a parent as key and its children as values.

Figure 4 shows the generated CHILD function (represented by a table) for AW source. We observe that, e.g. for node AW 1, we obtain the associated set of its children (here represented as an array) [Discipline 2, Curriculum 3, Course 4, Books 5].

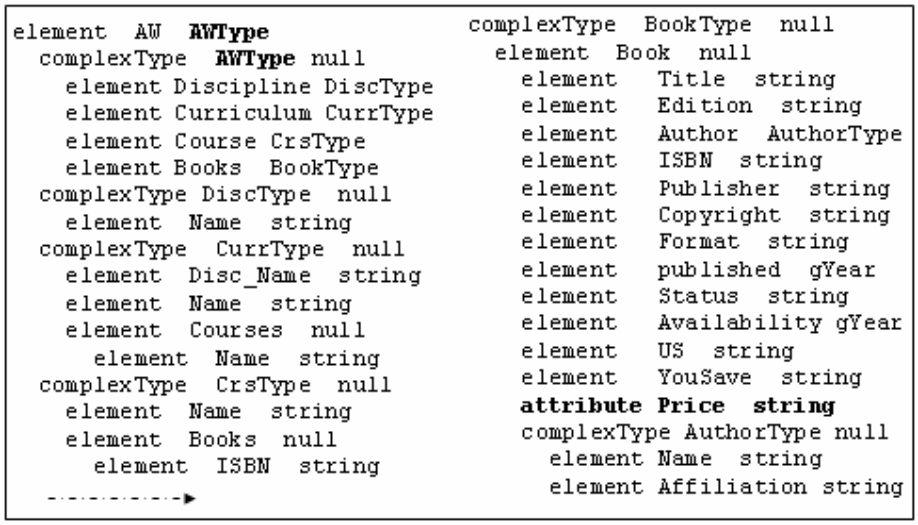

Figure 3. `Document object` structure for AW XSD.

```
{ Author 17    =   [ Name 28, Affiliation 29 ],
  Curriculum 3 =   [ Disc_Name 7, Name 8, Courses 9 ],
  Courses 9    =   [ Name 10 ],
  Course 4     =   [ Name 11, Books 12 ],
  AW 1         =   [ Discipline 2, Curriculum 3, Course 4, Books 5 ],
  Discipline 2 =   [ Name 6 ],
  Books 12     =   [ ISBN 13 ],
  Books 5      =   [ Book 14],
  Book 14      =   [ Title 15, Edition 16, Author 17, ISBN 18,
                     Publisher 19, Copyright 20, Format 21,  published 22,
                     Status 23, Availability 24, US 25, YouSave 26, @Price 27]
}
```

Figure 4. CHILD function table for source AW.

5. XSDs Mediation

In order to obtain local queries for a query issued against the global XML schema, the system must identify the XML data sources concerning a given query. For this task, the XML Metadata Document (XMD) is utilized as mediation to overcome the heterogeneity of data sources. XMD is proposed to maintain the correspondence between the components of the XSDs. For each component of the global schema, the objective is to keep the set of components having the same meaning in the local schemas and the semantic function if it is needed. Actually, since the global and the local schemas are trees, each node is identified by its path in the tree, called a *global path* for a component of the global schema and a *local path* for the corresponding component of a local schema. The relationship between a global path and a local path is assumed as a mapping. The distinction between components and paths is important, because a component may occur several times in an XSD structure with different meanings, while a path always identifies a unique

component. The correspondence among schemas is expressed through a set of mappings. These mappings capture the heterogeneity of the various data sources.

5.1 Mapping cases between XSD components

According to the number of nodes that are involved in the global XSD and a local XSD, mappings between their components are classified to One-to-One, One-to-Many, and Many-to-One. A component can be an element or an attribute. Several mapping cases are investigated in which conflicts may occur between components. In the next subsections, we describe some cases which are demonstrated above in Figure 2.

One-element to Many-elements this case can occur when there is a component represented as one element in the global XSD but as many elements in a local XSD. Hence, more than one element in a local XSD holds the same index number. Therefore to resolve this conflict a concatenation operation is needed for such a task. The <Format> in the global XSD is an example of this case.

Many-elements to One-element here more than one component in the global schema is represented by one component in a local schema. The <FName> and <LName> in the global XSD is an example, and a separate operation is needed.

One-attribute to One-element when a component is represented as an attribute in the global XSD and as an element in a local XSD, then the attribute name is just replaced by its equivalent element name.

One-attribute to One-attribute with a specific operation this is a case where a specific operation is required to resolve a semantic conflict among two components. For example, a conversion operation is needed to get the value of <@price> attribute in euro instead of dollars from the AW source.

5.2 XMD generation

In general, the major difficulty of connecting the global XML schema and the local XML schemas comes from the large number of data sources. Therefore, it is absolutely necessary to generate mappings automatically. The designer interaction is necessary; two terms may refer to different concepts and may not have the same meaning. Only a human at the present time is able to guarantee the semantic consistency of such a mapping. Hence, we implement a simple form (GUI) as an assistant tool for mapping generation. A part of a GUI is shown in Figure 6. The second column is used for assigning a unique index number for the equivalence paths. The third column is used to specify the function names which are needed to resolve heterogeneity conflicts by performing specific operations.

The process of XMD generation comprises the following steps:

1. The generated CHILD table for each XSD is traversed to obtain a unique path for each component of the XSD tree structure starting from the root.
2. A GUI is generated for each XSD.
3. Using the GUI for each XSD, a unique index number is assigned for the equivalent local and global paths.
4. In the third column of the GUI, either a null value is specified in the case of one-to-one mapping or the required function name is specified in other cases.

5. By collecting the same indices, the equivalent paths are grouped and the XMD document is easily created.

The XMD structure with its XSD is shown in Figure 5. Components in the global XSD are called *source components* <source>, while corresponding components in local XSDs are called *destination components* <dest>. In our example there are three local sources. Thus, each <source> element is followed by three <dest> elements. Moreover, XMD contains information about the required functions which is represented by the <function> element if it is needed to perform a specific operation for a specific <source> element.

```
<?xml version="1.0"  ?>
<Med_component>
   <source> "global.xml" </source>
      <dest> "source1.xml" </dest>
      <dest> "source2.xml" </dest>
      <dest> "source3.xml" </dest>
  <source> Titles/Books/Book/Format </source>
      <dest> AW/Books/Book/Format </dest>
      <dest> PH/Books/Book/Format </dest>
      <dest> Wiley/Books/Book/CoverType,
             Wiley/Books/Book/Pages </dest>
  <source> Titles/Books/Book/@price </source>
      <dest> AW/Books/Book/@price </dest>
      <dest> PH/Books/Book/price </dest>
      <dest> Wiley/Books/Book/price </dest>
    .......
</Med_component>
<Med_Func>
    ........
<source> Titles/Books/Book/Format </source>
      <function> null </function>
      <function> null </function>
      <function> Concatenate() </function>
<source> Titles/Books/Book/@price </source>
      <function> ToEuro() </function>
      <function> ToEuro() </function>
      <function> ToEuro() </function>
     ........
</Med_Func>
```

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<element name = "Med_component">
   <complexType>
      <sequence>
        <element name ="source" type = "string"/>
        <element name ="dest"   type = "string"/>
      </sequence>
   </complexType>
</element>
<element name ="Med_func">
   <complexType>
      <sequence>
        <element name ="source"   type = "string"/>
        <element name ="function" type = "string"/>
      </sequence>
   </complexType>
</element>
</schema>
```

Figure 5. A sample of an XMD XML document with its XSD.

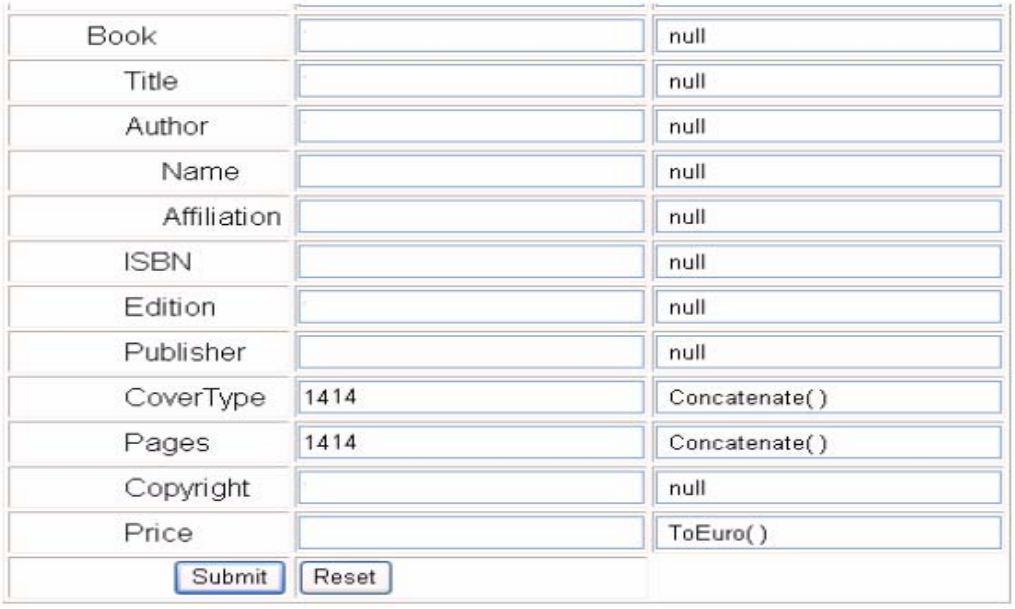

Figure 6. A part of the GUI for Wiley.

6. Query translation process

After the generation of the XMD, queries posed on the global XML schema can be evaluated. We developed a method to query the distributed heterogeneous XML data sources. A query translator unit is implemented, which is an integral part of the mediation layer. Its function is to translate global queries into queries suitable for the data sources. That is, if there is a correspondence between the paths in the global and local XSDs. When a global user query is posed, first it is parsed, then the XMD document is read, parsed by SAX, and the number of local sources is identified. The CHILD function is also used for the query translation process. A CHILD table *t* is constructed for the XMD, in which each <source> component value in XMD (global path) is represented as a key and associated with its <dest> components' values as values (local paths). Also <function> components' values in XMD are represented in *t* as values and their corresponding <source> value as key. For each path in the global query (should be a <source> component in XMD), if there is a non-empty value of the corresponding local components (<dest> component in XMD), then by navigating the XMD document, the paths in that query are replaced by paths to the <dest> values to get a local query. Otherwise, an empty query is generated for the corresponding path in the local query, which means this query cannot be applied to such local source. Each (generated) local query is sent to the corresponding local source engine, which will execute the query locally and return the result to the global query.

Algorithm: **Global query translation process**
Input: global XML query *q*, global XSD, and XMD document
Output: local XML queries q_1, q_2…, q_n
Step1: **parse** *q;*
Step2: **read** XMD, **identify** the number of local sources;
Step3: **construct** CHILD function *t* for XMD;
// *source components* as keys and *destination components* as values.
Step4: **for each** global path g_e in *q* **do**

materialize t;
for each source S_i in t having the corresponding local path l_e to g_e **do**
generate local query q_i for the first occurrence; //only once
for each g_e whose <function> value is not null
generate the required function operation
endfor
replace g_e by l_e in q_i;
endfor
endfor

Step5: **execute** the generated local queries locally.

7. Conclusion

In this paper, we have described an approach for resolving structural and semantic conflicts of heterogeneous XML data. We used XML Schema language for defining the XML data sources. A mediation layer is introduced to maintain the mappings among global and local XML schemas. Such layer consists of two main parts: the XMD and the Query Translator. The tree of each XML schema is constructed automatically and represented by a simple form to be used as a tool for assigning index numbers to all XSD component paths. A unique index number is assigned to nodes with the same meaning in order to resolve conflicts. The same index numbers are collected to generate each global path with its corresponding local paths. Then, the XMD is generated. Also, we have presented the second part of the mediation layer, the Query Translator. It acts to decompose global queries into a set of subqueries. A global query from an end-user is translated into local queries for XML data sources by looking up the corresponding paths in the XMD. Java 2, JDOM, JavaCC, and the Java servlet server were used as tools for the prototype implementation of this proposal.

Our implementation is still early naive prototype; many issues remain to be investigated. In the future, we plan to involve more features of XML Schema. For example, the current prototype does not support paths that contain wildcards. Removing redundancy will be also considered.

Acknowledgements. This work was supported in part by the National programme of research (Information society project 1ET100300419).

References

[1] FARQUHAR, A., FIKES R, and RICE J., The Ontiliqua Server: A tool for Collaborative Ontology Construction, in: International Journal of Human-Computer Studies, 1997, pp. 707-728.

[2] ALMARIMI, A., and POKORNY, J., Querying Heterogeneous XML data, in: Proc. on 6th Int. Baltic Conf. BalticDB&IS 2004, Riga, Latvia, pp. 177-191.

[3] LUDASCHER, B., GUPTA, A., and MARTONE, M. E., Model-based Mediation with Domain Maps, in: Proc. of Int. Conf. on Data Engineering, 2001, pp. 81-90.

[4] BARU, C., GUPTA, A., LUDASCHER, B, MARCIANO, R., PAPAKONSTANTINU, Y., VELIKHOV, P., and CHU, V., XML-Based Information Mediation with MIX, in: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 597-599.

[5] HAKIMPOUR, F., GEPPERT, A., Resolving semantic heterogeneity in schema integration: An ontology base approach, in: Chris Welty and Barry Smith (eds.), Proc. of Int. Conf. on Formal Ontologies in Information Systems, ACM Press, October 2001.

[6] WIEDERHOLD, G., Mediators in the Architecture of Future Information System, in: IEEE Computer Magazine, Vol. 25, No. 3, March 1992, pp. 38-49.

[7] ULLMAN, J., Information Integration Using Logical Views, in: Proc. of the Int. Conf. on Database Theory, 1997, pp. 19-40.

[8] HAAS, L., KOSSMAN, D., WIMMERS, E., and YOUNG J., Optimizing Queries across Diverse Data Sources, in: Proc. of 23rd Int. Conf. On Very Large Databases, Athens, Greece, 1997, pp. 276-285.

[9] LENZERINI, M., Data Integration: A Theoretical Perspective, in: Proc. of the ACM Symposium on Principles of Database Systems, Madison, Wisconsin, USA, June 2002, pp. 233-246.

[10] SAX 1.0: The Simple API for XML. http://www.perfectxml.com/wp/3110_Chapter06/contents.htm

[11] MAY, W., A Rule-Based Querying and Updating Language for XML, in: Proc. of the Workshop on Databases and Programming Languages, Springer LNCS 2397, 2001, pp. 165-181.

[12] W3C Consortium: Extensible Markup Language (XML). http://www.w3.org/TR/2000/REC-xml

[13] W3C Consortium: XML Schema Part 0 : Primer. http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[14] W3C Consortium: XML Schema Part 1: Structures. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

[15] W3C Consortium: XML Schema Part 2: Datatypes. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[16] NAM Y., GOGUEN, J., WANG, G., A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases, in: Proc. of the Confederated International Conferences DOA, CoopIS and ODBASE, Irvine CA, October 2002, LNCS 2519, Springer, pp. 1332-1344.

[17] PAPAKONSTANTINOU, Y., GARCIA-MOLINA, H., ULLMAN, J., MedMaker: A Mediation System Based on Declarative Specifications, in: Proc. of the IEEE Int. Conf. on Data Engineering, New Orleans, LA, February 1996, pp. 132-141.