XML DATA INTEGRATION SYSTEM

Abdelsalam Almarimi

The Higher Institute of Electronics Engineering
Baniwalid, Libya
Belgasem_2000@Yahoo.com

ABSRACT

This paper describes a proposal for a system for XML data Integration and Querying via Mediation (XIQM). An XML mediation layer is introduced as a main component of XIQM. It is used as a tool for querying heterogeneous XML data sources associated with XML schemas of diverse formats. Such a tool manages two important tasks: mappings among XML schemas and XML data querying. The former is performed through a semi-automatic process that generates local and global paths. An XML Query Translator for the latter task is developed to translate a global user query into local queries using the mappings that are defined in an XML document.

KEYWORDS: Data integration, mediation, XML Schema, XML Query languages.

1. INTRODUCTION

XML [10] has become 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 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 [11] provides a notation for defining new types of XML elements and XML documents.

Our system prototype called XIQM (XML data Integration and Querying via Mediation) has been built to perform the mappings among XML schemas, producing a mediation layer, which is then used to generate local queries. The mediation layer is proposed as a main component 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. 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. Thus, we implement a simple form (GUI) as an assistant tool for mapping generation.

In the paper [1], only one unit of our system was introduced. We described the process of the XMD generation in which a semi-automatic process for integrating XML schemas is implemented using GUI. Such a process takes a set of local XML schemas and generates an XML Metadata Document that can be used for query purposes. In this paper, we introduce the whole system including a new method for XML query translation which is the integral part of the system. Currently, we use Quilt [3] as XML query language, but we can move to XQuery language without problems.

The rest of the paper is organized as follows. Section 2 introduces the related work. In section 3, we present an overview of the system architecture. Section 4 describes the schema integration process. Section 5 shows the XML Metadata Document generation. Section 6 describes the query translator unit. Examples of query translation are introduced in Section 7. Finally, we conclude the paper and point out the future work.

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 [9]. In this architecture, mediators provide a uniform user interface to views of heterogeneous data sources. They resolve queries over global concepts into sub-queries over data sources.

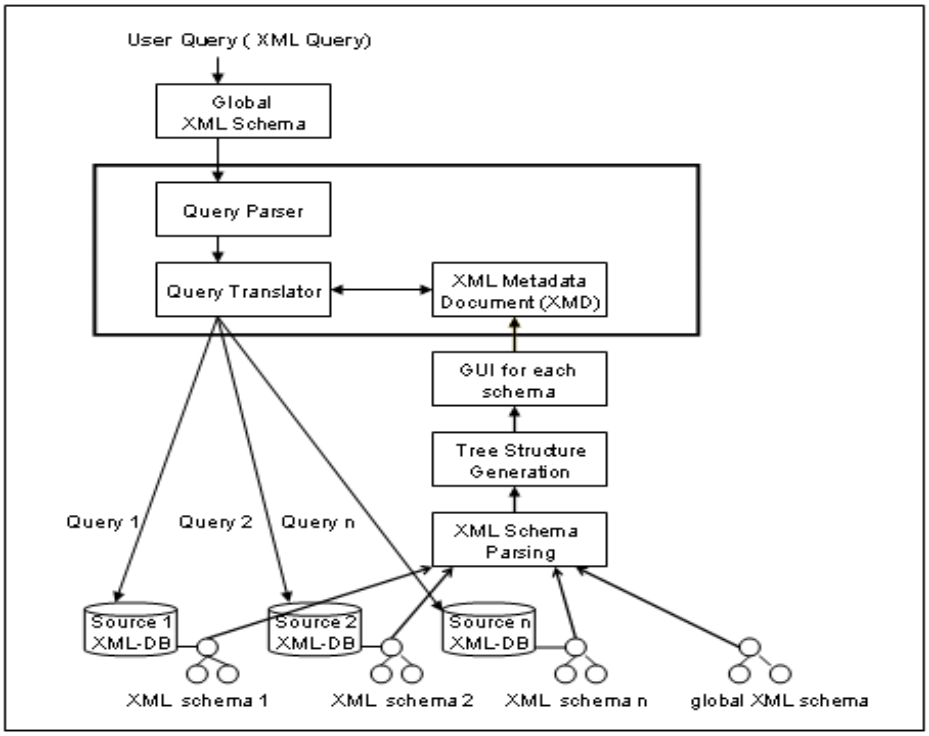

Figure 1: The system architecture.

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 [5]. 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 integration system without the need to restructure the global schema.

Well-known research projects such as Garlic [4], Tsimmis [8], MedMaker [7], and Mix [2] are structural approaches. 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 [6] (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 [5].

We classify our system as a structural approach and differ from the others by following both the local-as-view and the global-as-view approaches. In addition to this, the XML Schema language is adopted in our work instead of DTD grammar language, which has limited applicability.

3. SYSTEM ARCHITECTURE

The entire architecture of XIQM is presented above in Figure 1. The data sources that we are interested in are XML documents satisfying different XML schemas. The XML Schema language is adopted in our work instead of DTD (Document Type Definition) grammar language, which has limited applicability. 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. A GUI assistant tool is also involved, which is a simple form used to simplify the mapping process among schemas and to reduce the designer effort.

The function of the query translator is rewriting a parsed global query into a local query for each local source. The main idea is that when a global query over the global XML schema is posed, it is automatically translated by the Query Translator unit into sub-queries, called local queries, which fit each local database format using the information stored in XMD.

4. AN APPLICATION EXAMPLE

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 (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 PH 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.

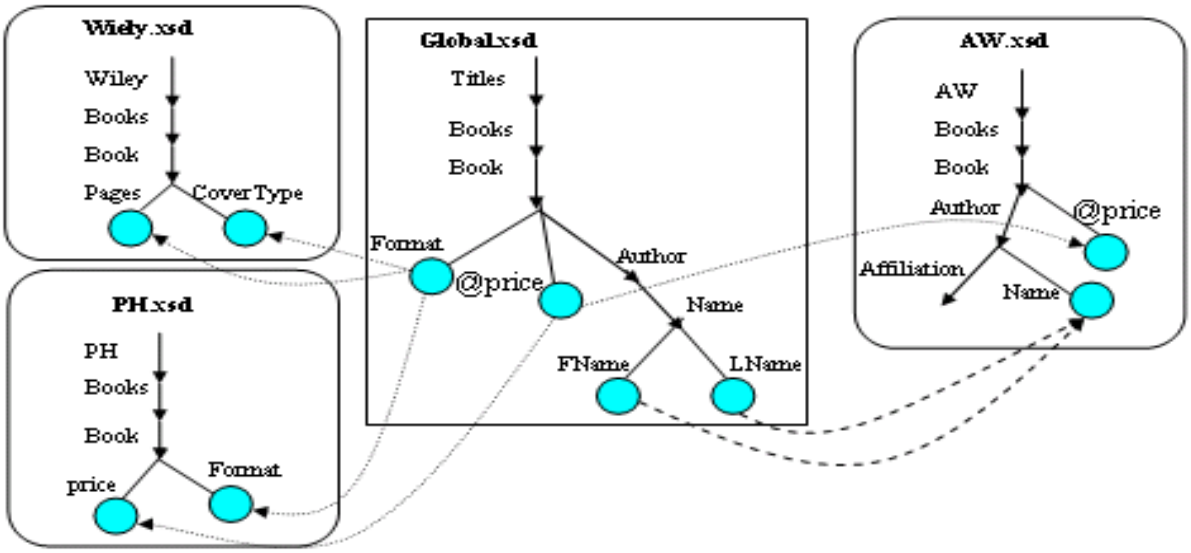

Figure 2: A Part of the tree structure for XSDs.

5. XML SCHEMA PROCESSING

The XML schema is itself an XML document, which we denote as XSD (XML schema definition). 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 XML schema documents in memory. In fact, JDOM itself does not include a parser. Instead it depends on a SAX parser [12], which can be used to parse documents and build JDOM models from them.

5.1 XSD MODELING

We model XSD as a tree structure whose nodes are components of the corresponding local sources.

```
{ 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 3: CHILD function table for source AW.

Each component corresponds to the occurrence of a tag, to the occurrence of an attribute, to the content of tag, and so on. We formulate an XSD Model (XModel), where an XSD can be described. We consider a set of nodes N that can be represented as: E for element names, A for attribute names, and T for type names. Here, we do not aim at complete formalization of all details of XML Schema, but rather capturing its essential modeling features which meet the basic requirements of our approach. Hence, we consider the below functions as basic requirements in which an XSD can be characterized:

Root: Φ → N returns the unique root node of the document,
Children: N → [N] returns the ordered list of children of a node, or the empty list [ ] in the case of a leaf node,
Attr: N → A returns an attribute name,
Tag: N → E returns the unique tag (e.g. element name) of a node,
Type: N → T returns a type name of a node.

Definition 5.1.1 (Names)
An XModel schema is a tree that holds a set of nodes N which can be disjoint sets T, E, and A of type names, element names, and attribute names, respectively.

Definition 5.1.2 (Types)
Given an XModel for an XML schema definition, then each type $t \in T$ in the schema is either a simple type Ts, or a complex type Tc, where $T = Ts \cup Tc$.

5.2 EXTRACING XSD COMPONETS

To generate a unique path for each element or attribute of the schemas, we need to search the XModel structure and extract out the components that we are interested in. Mainly, only the values of the elements and attributes are requested. In other words, we omit the names of components which contain a complex type. Therefore, we assume that ELEMENTS and ATTRIBUTES are a set of elements and attributes, respectively, of the XModel. Formally, we introduce a function:

$$\text{CHILD:COMPONENT} \rightarrow \wp(\text{COMPONENT}),$$
$$\text{COMPONENT} = \text{ELEMENTS} \cup \text{ATTRIBUTE}$$

which assigns a multi set of child components to each component in an XSD1. Basically, the CHILD function is founded to materialize the XSD components that are needed. The process of extracting XSD components comprises the following steps:

1. After a XModel tree 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 given to each node of x to resolve naming conflicts.
4. A depth-first traversal is performed on x; and the CHILD function is materialized. Figure 3 shows the generated CHILD function (represented by a table) for AW source. It is obvious 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].

```
<?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>
```

Figure 4: A sample of an XMD document.

5.3 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. 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.
5. A simple form (GUI) is generated for each XSD as an assistant tool for mapping generation.
6. Using the GUI for each XSD, a unique index number is assigned for the equivalent local and global paths. Also, a null value is specified in the

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

case of one-to-one mapping or the required function name is specified in other cases. By gathering the same indices, the equivalent paths are grouped and the XMD document is easily created.

A sample of XMD structure with its XSD is shown in Figure 4. 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.

6 QUERY TRANSLATION PROCESS

Once the XMD is generated, queries can be posed on the global XML schema and easily evaluated. We developed a method for querying 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 that are fit to the data sources. When a global user query is posed against the global schema, first it is parsed, then The XMD document is read, and also parsed by SAX, and the number of local sources is identified. The same idea which we have used for XML schema parsing is also applied here. In this case the CHILD function is of the form

$$\text{CHILD: SOURCE} \rightarrow \wp\,(\text{DESTINATION}) \cup \wp\,(\text{OPERATION})$$

where the SOURCE is a set of the global paths in the XMD, the DESTINATION is denoted to the corresponding local paths, and the OPERATION is a set of the corresponding semantic functions' names which are used for resolving heterogeneity conflicts. A CHILD function table t is constructed for the XMD, in which each <source> element value in XMD (global element path) is represented as a key and associated with their <dest> elements' values as values (local elements' paths). Also <function> elements' values in XMD are represented in t as values and its corresponding <path> value as key.

The basic idea is that, 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.

7 QUERY TRANSLATION EXAMPLES

In order to support our analysis, we introduce some examples of XML queries. Two cardinality cases are investigated: many-to-one and one-to- many.

7.1 MANY-TO-ONE QUERY EXAMPLE

```
Q1:
  FOR $a IN   document("global.xml")//Author//Name
   RETURN
    <Author>
      <Name> $a/LName, $a/FName </Name>
    </Author>
```

In this example, the LName and FName elements in the global schema are mapped to Name element in a local schema. That means we need to provide a specific function to separate the full name into a first name and last name. From the XMD fragment in Figure 4, it is clear that two different <source> element values (in XMD document) have the same <dest> element values. In other words, we need to split the instance value of the destination element value to generate the appropriate local queries. The resulted local query for AW source as following:

```
FUNCTION Fname_fun ($par)
{split (" ", $par) [1]}

FUNCTION LName_fun ($par)
{document (split (" ", $par) [2])}

FOR $b IN document("source1.xml")//Book
RETURN
 <Book>  FName_Fun($a/name),
         LName_Fun($a/name)
</Book>
```

7.2 ONE-TO-MANY QUERY EXAMPLE

```
Q2:
  FOR $b IN  document("global.xml")//Book
  RETURN
    <Book>$b/Format</Book>
```

This example illustrates the case that can be happened when there is a node in the global schema mapped to many nodes in a local schema. Figure 2 above describes the mapping of this query. To answer this kind of mapping we need to provide a specific function to perform this task. For example, the global element Format is represented in Wiley by two different local elements: Pages and CoverTypes. Figure 5 shows the execution of Q2.

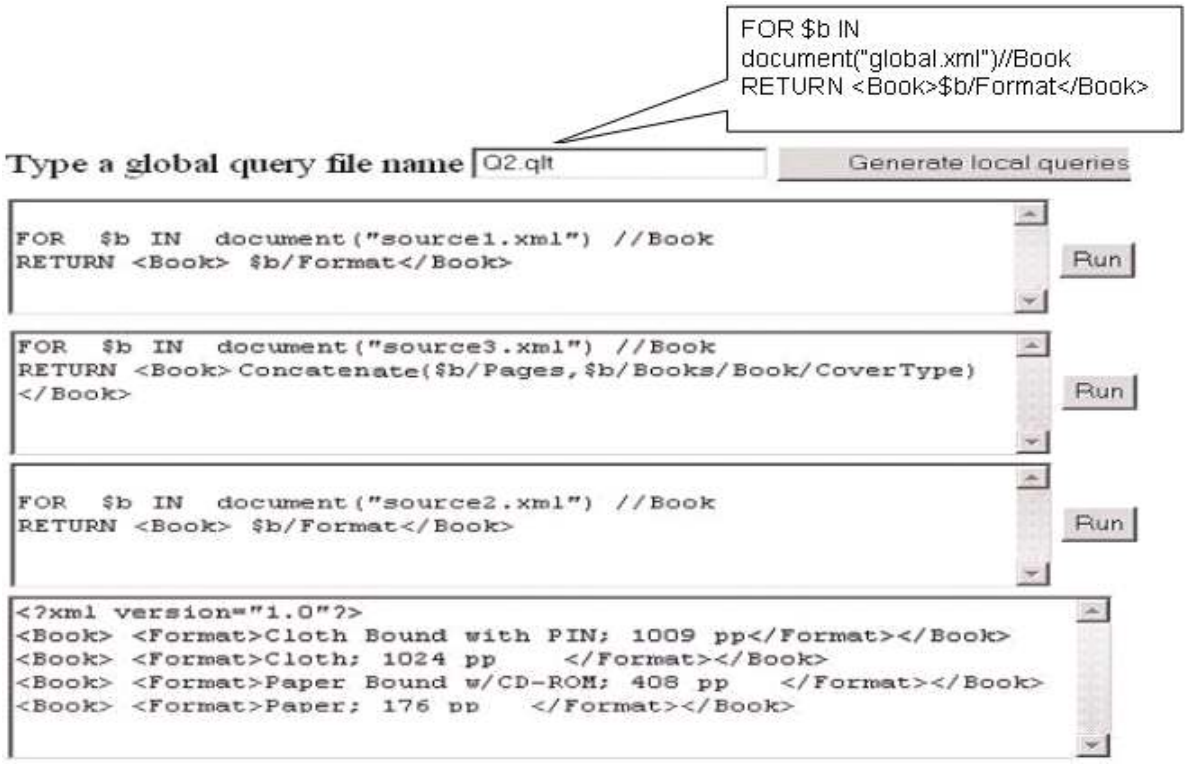

Figure 5: Example of a global query translation.

8 CONCLUSIONS

In this paper, we have described our system for resolving structural and semantic conflicts for distributed heterogeneous XML data. We developed a mediation layer for querying heterogeneous distributed XML data sources. This layer holds two main parts: the XMD and the Query Translator. We used XML Schema language for defining the XML data sources. The mediation layer is used for describing the mappings between global and local schemas. XML schemas' trees are generated automatically, each with a GUI. Semantic discrepancies are resolved by using the GUI tool for assigning index numbers to all database elements' paths.

Also, we have presented the second part of the mediation layer, the Query Translator. It acts to decompose global queries into a set of sub-queries. 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 achieved. In the future, we plan to involve more features of XML Schema. For example, the current prototype does not support paths that contain wildcards. Also, we plan to move to XQuery, instead of Quilt.

REFERENCES

[1] Almarimi A., and Pokorny J., "A Mediation Layer for Heterogeneous XML schemas," *The International Journal of Web Information Systems (IJWIS)*, vol. 1, no. 1, pp. 25-32, 2005.

[2] Baru C., Gupta A., Ludascher, B Marciano R., Papakonstantinou Y., Velikhov P., and Chu V., "XML-Based Information Mediation with MIX," i*n Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 597-599, 1999.

[3] Chamberlin D., Robie J., and Florescu D., "Quilt: An XML Query Language for Heterogeneous Data Sources," *in ACM SIGMOD Associated Workshop on the Web and databases*, Dallas, Texas, pp. 53-62, 2000.

[4] Haas L., Kossman D., Wimmers E., and Young J., "Optimizing Queries across Diverse Data Sources," *in: proceedings of 23rd Int. Conf. On VLDB*, Athens, Greece, pp. 276-285, 1997.

[5] Lenzerini M., "Data Integration: A Theoretical Perspective," *in Proceedings of the ACM Symposium on Principles of Database Systems*, Madison, Wisconsin, USA, pp. 233-246, 2002.

[6] Nam Y., Goguen J., Wang G., "A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases," *in Proceedings of the Confederated International Conferences DOA, CoopIS and ODBASE*, Irvine CA, LNCS 2519, Springer, pp. 1332-1344, 2002.

[7] Papakonstantinou Y., Garcia-Molina H., Ullman J., MedMaker: "A Mediation System Based on Declarative Specifications," *in Proceedings of*

the IEEE Int. Conf. on Data Engineering, New Orleans, LA, pp. 132-141, 1996.

[8] Ullman J., "Information Integration Using Logical Views" *in Proceedings. of the Int. Conf. on Database Theory*, pp. 19-40, 1997.

[9] Wiederhold G., "Mediators in the Architecture of Future Information System," *in IEEE Computer Magazine*, vol. 25, no. 3, pp. 38-49, 1992.

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

[11] W3C Consortium: XML Schema, www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[12] SAX 1.0, The Simple API for XML, www.perfectxml.com/wp/3110_Chapter06/contents.htm