

# Establishing an XML Metadata Knowledge Base to Assist Integration of Structured and Semi-structured Databases

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

Department of Computer Science  
Cardiff University  
Wales, UK

{Wasil, W.A.Gray, N.J.Fiddian}@cs.cardiff.ac.uk

## Abstract

This paper describes the establishment of an XML Metadata Knowledge Base (XMKB) to assist integration of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents (XML documents that conform to the XML syntax rules but have no referenced DTD or XML schema) produced by internet applications. We propose an approach to combine and query the data sources through a mediation layer. Such a layer is intended to establish and evolve an XMKB incrementally to assist the Query Processor to mediate between user queries posed over the master view and the distributed heterogeneous data sources. The XMKB is built in bottom-up fashion by extracting and merging incrementally the metadata of the data sources. The XMKB is introduced to maintain the 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. A System to Integrate Structured and Semi-structured Databases (SISSD) has been built that generates a tool for a meta-user (who does the metadata integration) to describe mappings between the master view and local data sources by assigning index numbers and specifying conversion function names. This system is flexible: users can get any master view from the same set of data sources depending on their interest. It also preserves local autonomy of the local data sources. 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 and to regenerate the XMKB from scratch.

*Keywords:* Well-formed XML document, Relational database, XML queries, semi-structured data, semantic mapping, data integration.

## 1 Introduction

With the growth, widespread and increasing popularity of the Internet the number of data sources available for public access is rapidly increasing both in number and size, while at the same time users and application programs increasingly need to combine data from these different autonomous and heterogeneous data sources (Segev and Chatterjee, December 1991.) (Karunaratna et al., May 1998). However, for the foreseeable future, most data will continue to be stored in relational database systems because of the reliability, scalability, tools and performance associated with these systems (Funderburk et al., 2002) (Shanmugasundaram et al., September 2000). Additionally, many web-based applications and web services publish their data using XML (Lehti and Fankhauser, 2004), therefore much interesting and useful data can be found in well-formed XML documents. Hence, building a data integration system that provides unified access to semantically and structurally diverse data sources is very desirable to link structured data held in relational databases and semi-structured data in XML documents (Gardarin et al., 1999, Lee et al., 2002). 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 (Kurgan et al., 2002). 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. 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 for authors' or editors' names (e.g., full name or separated first name and last name), or different units for prices. 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 can not provide enough 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, where mediation

engineers are given an easy tool to describe mappings between the integrated (integrated and master are used interchangeably in this paper) schema and local schemas, to produce a uniform view over the local databases (Young-Kwang et al., October 2002).

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 the best 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.

This paper mainly refers to the problem of integrating 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. These XML documents can be XML files on local hard drives or remote documents on web servers. We propose an approach to combine and query the data sources through a mediation layer. Such a layer is intended 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 distributed heterogeneous data sources, to translate such queries into sub-queries -called local queries- which fit each local data source, and to integrate the results. The XMKB is built in a bottom-up fashion by extracting and merging incrementally the metadata of the data sources. The XMKB is an XML document which includes the database or XML document name, type and location information, and the metadata, in which the mappings between the master view and schema structures of the data sources are defined. A System to Integrate Structured and Semi-structured Databases (SISSD) has been built that generates a tool for meta-users to do the metadata integration, producing an XML Metadata Knowledge Base (XMKB), which is then used to generate queries to local data sources from user queries posed over the master view, and to integrate the results. This tool parses the master view to generate automatically an index number for each element and parses local schema structures to generate a path for each element, and produce a convenient GUI. The mappings assign indices to match local elements to corresponding master elements and to names of conversion functions. These functions can be built-in or user-defined functions. The XMKB is then generated based on the mappings by combination over index numbers. User queries are expressed in FLWR expressions of XQuery (a powerful universal query language for XML) and processed

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

This system is flexible: users can get 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 any 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 any data sources are added to or removed from the system without the need to regenerate it from scratch.

The rest of the paper is organized as follows. The next section presents related work. The architecture and its main components are described in section 3. Section 4 presents the structure, content and the organization of knowledge in the XMKB and how it is generated. Finally, we present conclusions in section 5.

## 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 (Wiederhold, March 1992). In this architecture, mediators provide a uniform user interface to query integrated views of heterogeneous data sources. They resolve queries over global concepts into sub-queries over data sources. Mainly, they can be classified into structural approaches and semantic approaches.

In structural approaches, local data sources are assumed to be 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 achieved by sharing a common ontology among the data sources. According to the mapping direction, the approaches are further classified into two categories: global-as-view and local-as-view (Lenzerini, 2002). 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 is well-suited to supporting a dynamic environment, where data sources can be added to or removed from the data integration system without the need to restructure the global schema.

There are several well-known research projects and prototypes such as Garlic (Carey et al., 1995), Tsimmis (Ullman, 1997), MedMaker (Papakonstantinou et al., 1996) and Mix (Baru et al., 1999) which take a structural and 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

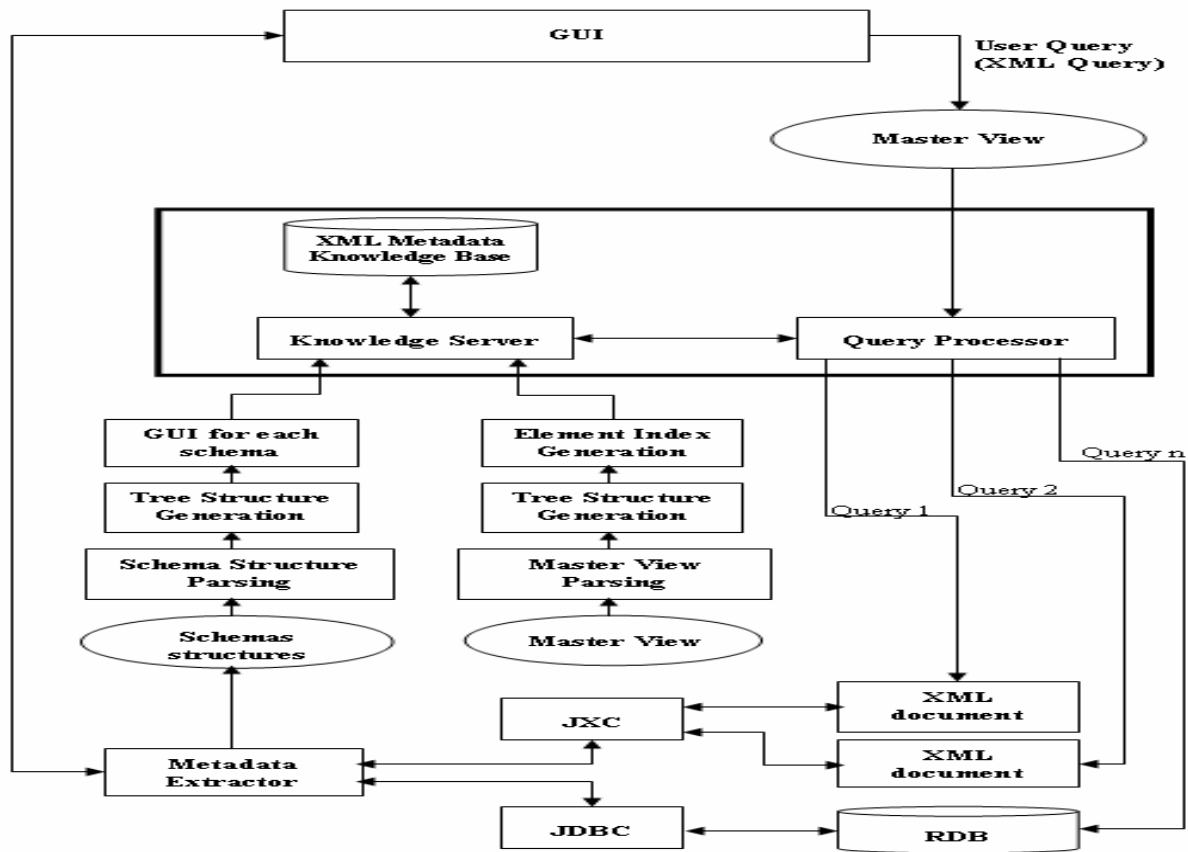


Figure 1: The Architecture of our System.

the view definition language there. DDXMI (Young-Kwang et al., October 2002, Nam et al., 2003) (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 based on the DTD parsing that is associated with each source. (Almarimi and Pokorny, 2004) describe an approach for mediation of heterogeneous XML data sources. Their approach is proposed as a tool for an XML data integration system to combine and query XML documents through a mediation layer. This layer is intended to describe the mappings between the 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. In this case the XML data sources are described by the XML Schema language.

We classify our work as being in the structural category but we differ from the others (Ullman, 1997, Papakonstantinou et al., 1996, Baru et al., 1999) by following the local-as-view approach. The XML documents that we are interested in are well-formed XML documents which have no referenced DTD or XML schema, while the work in (Young-Kwang et al., October 2002, Nam et al., 2003) is interested in XML documents designed according to DTD definitions, and that in (Almarimi and Pokorny, 2004) is interested in XML documents satisfying different XML schemas. Also our work differs from the others (Young-Kwang et al., October 2002; Nam et al., 2003; Almarimi and Pokorny, 2004) by using an incremental tool to build the XML Metadata Knowledge Base (XMKB). This tool would start from the previous XMKB file and slightly modify it in light of slight modifications to data sources schema structure or when any data sources are added to or removed from the system, instead of regenerating it from scratch.

### 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)

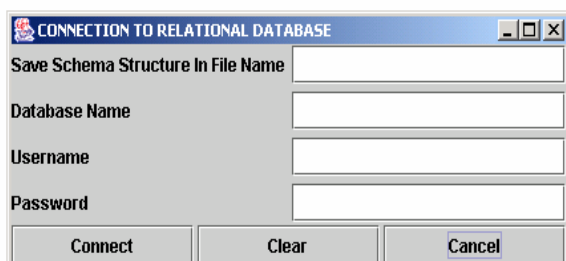
The MDE needs to deal with heterogeneity at the hardware, software and data model levels without violating the local autonomy of the data sources. It interacts with the data sources via JDBC (Java Database Connectivity) if the data source is a relational database or via JXC (Java XML Connectivity) if 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 have 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

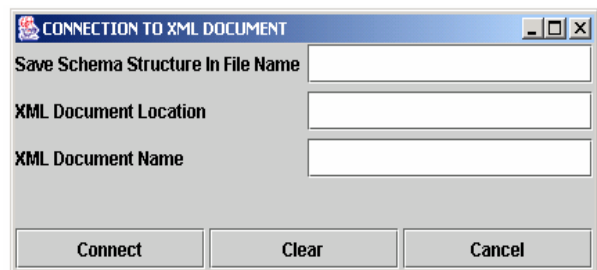
Typically, the heterogeneous data sources use different data models to store the data (e.g. relational model and XML model). This type of heterogeneity is referred to as syntactic heterogeneity. The solution commonly adopted to overcome syntactic heterogeneity is to use a common data model and to map 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 generates a 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 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 receives a user query over the master view to process it and returns the query result to the user in integrated form. User queries are expressed in XQuery (a powerful universal query language for XML) using FLWR expressions. XQuery offers seven types of expression, but FLWR expressions are among the most interesting types of expression it offers. Using these expressions for queries over the 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 Query Processor is composed of several components in charge of:

- Rewriting the user query into sub-queries -called local queries- which fit each local data source, by using the mapping information stored in the XMKB.
- Converting the XQuery sub-queries addressed to the relational databases into SQL queries to execute them, then converting the results into XML format.
- Sending local queries to their corresponding local data source engine, to process the query and return the results.
- Merging the results of the sub-queries.

## 4 The XML Metadata Knowledge Base (XMKB)

The building of the XML Metadata Knowledge Base (XMKB) is performed through a semi-automatic process. The XMKB is generated based on mappings between the master view and the local schemas, and includes the data source information (names, types and locations), XML path information (a path for each node starting from the root), and semantic information about XML elements (function names to resolve structural and semantic conflicts).

### 4.1 The Structure of XMKB

The XML Metadata Knowledge Base (XMKB) is an XML document composed of two parts. The first part contains information about data source name, type and location. 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 with its schema is shown in Figures 2 and 3, respectively. The *DS\_information* element in Figure 2 contains data source names, types and locations. The *DS\_information* element has one attribute called *number* which holds the number of data sources present in the integration system (3 in the example shown). Also the *DS\_information* element has child elements called *DS\_Location* elements. 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 the data source type is XML document) and the data source location that the system will submit the generated sub-query to. The *Med\_component* element in Figure 2 contains the 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 elements are called *source* elements, while corresponding elements 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. Also the source elements in this document have child elements called *target* which contain the corresponding path for the master view elements 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.

```
<?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>
```

Figure 2: A sample XMKB document.

### 4.2 How to Generate an XMKB

The XML Metadata Knowledge Base (XMKB) is utilized in mediation to overcome the heterogeneity of data sources. XMKB is intended to maintain the correspondence between the components of the data sources. For each component of the master view, the

objective is to record the set of components having the same meaning in the local schema structures and the discrepancy resolution function if it is needed.

Each data source (relational database or well-formed XML document) has its own schema structure in XML format constructed by the Meta-data Extractor (MDE). We assume that elements in local data sources do not contain attributes. This implies that data source schema structures can be represented as n-ary trees. Our approach involves mapping paths in the master view to (sets of) paths in the local schema structures, though we often speak of elements instead of the paths that lead to these elements. We match an element in the master view with elements in local data source schema structures, through generating an index number for each element in the master view tree and then assigning these index numbers

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="XMKB">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DS_information">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="DS_Location" maxOccurs="unbounded">
                <xs:complexType mixed="true">
                  <xs:attribute name="name" type="xs:string" use="required" />
                  <xs:attribute name="type" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="number" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Med_component">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="source" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="target" maxOccurs="unbounded">
                      <xs:complexType mixed="true">
                        <xs:attribute name="name" type="xs:string" use="required" />
                        <xs:attribute name="fun" type="xs:string" use="required" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="path" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 3: An XML schema of an XMKB document.

to the element(s) with the same meaning in the local schema structure trees. Hence elements with the same number have the same meaning. By collecting all elements with the same numbers, the source and target paths can be generated automatically, and the XMKB can be easily constructed. An especially convenient special case is where an element in the master view exactly matches one in a local schema structure, in that its field has the same meaning as the one in the master view. Elements in local schema structures should not appear in the XMKB file if their meaning does not relate to any element in the master view.

Constructing an XMKB file manually is an error prone and tedious job, so that machine support is highly desirable. Hence, we have developed a system that constructs an XMKB automatically. We implement a simple form (GUI) -from parsing a local schema structure- as an assistant tool for mapping generation. For

example, Figure 4 presents part of a GUI for the local schema structure shown in Figure 5.

Figure 4: A GUI for local schema structure of the bib XML document.

The first column is used for assigning the unique index numbers of master view elements to the equivalent elements in the local schema structure. Elements without an equivalent index number are not included in the XMKB document. The second column is used to specify the function names which are needed to resolve heterogeneity conflicts by performing specific operations.

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema_information>
  <data_source_information>
    <name>bib.xml</name>
    <location>C:\prototype\doc</location>
    <type>XML document</type>
  </data_source_information>
  <structure>
    <element name="bib">
      <element name="book">
        <element name="title" />
      </element>
      <element name="author">
        <element name="last" />
        <element name="first" />
      </element>
      <element name="editor">
        <element name="last" />
        <element name="first" />
        <element name="affiliation" />
      </element>
      <element name="publisher" />
      <element name="price" />
    </element>
  </structure>
</schema_information>
```

Figure 5: Schema structure of the bib XML document.

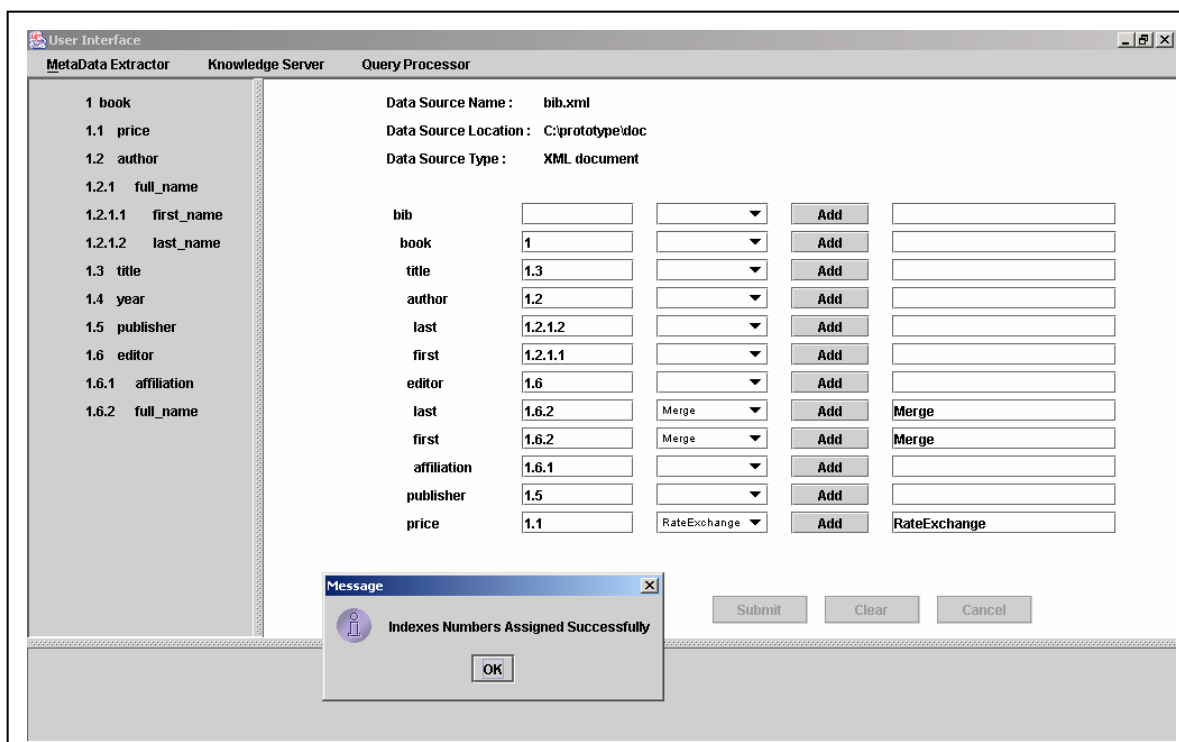


Figure 6: Example of mapping between master view and XML document.

The process of XMKB generation comprises the following steps:

1. Automatically generate unique index numbers for the master view elements.
2. Produce a convenient GUI for each local schema structure.
3. Using the GUI for each local schema structure, the unique index numbers of the master view elements are assigned to the equivalent local schema structure elements. Figure 6 shows an example of mapping between a master view (on the left of the figure) and the schema structure of the bib XML document shown in Figure 5.
4. Use the second column of the GUI form to specify the function names which are needed to resolve any heterogeneity conflicts by performing specific operations.
5. After assigning index numbers and function names, mapping paths in the master view to (sets of) paths in the local schema structures are generated for each element starting from the root, see Figure 7.
6. Finally, data source information (name, type and location), mapping paths and function names needed to resolve semantic and structural conflicts are merged with the XML Metadata Knowledge Base (XMKB) based on the mappings by collecting elements with the same index numbers.

This 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. The SISSD gives the flexibility to remove any data source schema from the XMKB and then add this data source again with an updated or altered schema without any other impact on the XMKB or the need to regenerate it from scratch.

#### 4.2.1 Index number generation for the master view elements

The generated index numbers for the master view elements are used to match local elements to corresponding master elements. We employ a mechanism to generate such index numbers using JDOM technology. By applying this mechanism, a unique index number is generated for each element in the XML document whatever the nesting complexity of the document.

The process of generating the index numbers comprises the following steps:

1. First, use JDOM technology to read and parse the XML document (master view) and map it to a tree.
2. Identify the root element of the document and assign index number 1 to it.
3. For each element in the document (including the root element), get all the children of this element.

Integrated Schema Element path	Data Source Element path	Mapping Function
/book	/bib/book	
/book/price	/bib/book/price	RateExchange
/book/author	/bib/book/author	
/book/author/full_name	Null	
/book/author/full_name/first_name	/bib/book/author/first	
/book/author/full_name/last_name	/bib/book/author/last	
/book/title	/bib/book/title	
/book/year	Null	
/book/publisher	/bib/book/publisher	
/book/editor	/bib/book/editor	
/book/editor/affiliation	/bib/book/editor/affiliation	
/book/editor/full_name	/bib/book/editor/last, /bib/book/editor/first	Merge

Figure 7: Generated mapping paths.

4. Assign a sequential number starting from 1 for each child to represent the order of children for that parent.
5. Combine this number given for the child with the index number of the parent separated by dot (.) and this will be the index number of that child. For example, if the root element has four child elements, the index number of the first child element will be 1.1, the index number of the second child element will be 1.2, and so on.

#### 4.2.2 Mapping between elements

According to the number of elements that are involved in the master view and a local schema structure, mappings between them are classified as One-to-One, One-to-Many or Many-to-One. For example, a local data source may represent author names as full names, while the master view separates the first and last names. In this case, the answer from the local data source must be split up if a query is to retrieve the first name of the author. Several mapping cases are possible in which such conflicts may occur between elements. In the next subsections, we describe some cases.

- **One to One with semantic functions:** this can occur when an element in the master view matches one element in a local schema structure but they use different reference systems. For example, the master view may represent price elements in dollar currency, while the local data source uses sterling currency or represents prices in cents. Therefore to resolve this conflict some conversion mechanism is required to translate between such representations.

- **One-element to Many-elements:** this case can occur when there is one element in the master view mapped to many elements in a local schema structure. Hence, more than one element in the local schema structure has the same index number. For example, the master view may represent an editor name as a full name, while the local data source separates an editor's first and last names. Therefore to resolve this conflict we need a function to concatenate the first and the last name elements to get the full name.
- **Many-elements to One-element:** this case can occur when there is more than one element in the master view corresponding to one element in a local schema structure. Hence, the element in the local schema structure will have more than one index number and more than one function name. For example, the master view may represent an author name as first\_name and last\_name, while the local data source represents it as a full name. Therefore to resolve this conflict two functions, *firstName* and *lastName*, are needed to split the author's full name into separate first name and last name. Figure 8 shows an example of Many-elements to One-element mapping and how this is done inside the GUI.

## 5 Conclusions

In this paper, we have described an approach for establishing an XML Metadata Knowledge Base (XMKB) to resolve structural and semantic conflicts between distributed heterogeneous structured data residing in relational databases and semi-structured data

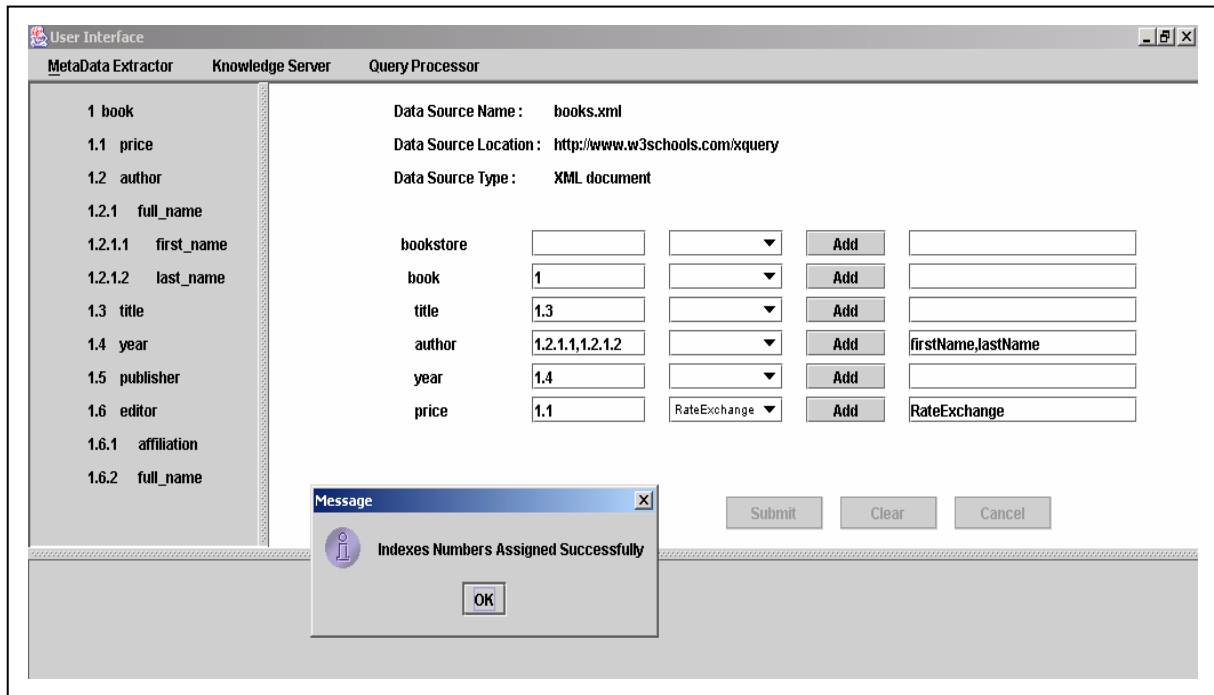


Figure 8: Example of Many-elements to One-element mapping.

held in well-formed XML documents produced by internet applications. The XMKB is employed to maintain the 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. A GUI tool is generated automatically for a meta-user (who does the meta-data integration) to describe mappings between the master view and local data sources by assigning a unique index number generated automatically for master view elements to the element(s) with the same meaning in the local schema structures, and specifying any necessary conversion functions for resolving structural and semantic conflicts. The XMKB is built based on these mappings by collecting element paths with the same index numbers to contain information about data structures and semantics. It can then be used by the Query Processor (QP) to mediate between user queries posed over the master view and the distributed heterogeneous data sources, to translate such queries into sub-queries -called local queries- which fit each local data source, and to integrate the results of these sub-queries.

The SISSD has been developed using Java, JDOM, and the JavaCC compiler. We have implemented the Meta-data Extractor (MDE) using JDBC and JDOM technology. We use JDBC as the API to connect to a relational database system. As a result, our implementation works on top of most commercial database systems including DB2, Oracle and Microsoft SQL Server, and on most hardware platforms. We have developed JXC using a JDOM (Java Document Object Model) interface to detect and extract the schema structure of a well-formed XML document, where the metadata are buried inside the data. The SISSD also preserves local autonomy of the local data sources, thus

any data sources can be handled without rebuilding or modifying the XMKB.

Certain issues remain to be investigated. For example, if some elements in the local data sources contain attributes and these attributes correspond to elements in the master view, how mapping between them would be achieved. This is not yet implemented, but should not be difficult. There are other matters, notably relating to the SISSD Query Processor, which have been outlined but not discussed in any great detail in this paper: it is intended that these will be elaborated in other, separate, publications.

## 6 References

- W3C Consortium: Extensible Markup Language (XML). <http://www.w3.org/TR/2000/REC-xml>.
- ALMARIMI, A. & POKORNY, J. (2004) A Mediation Layer for Heterogeneous XML Schemas. *Proceedings of the Sixth International Conference on Information Integration and Web Based Applications & Services (iiWAS2004)*. Jakarta, Indonesia.
- BARU, C., GUPTA, A., LUDÄSCHER, B., MARCIANO, R., PAPAKONSTANTINOU, Y., VELIKHOV, P. & CHU, V. (1999) XML-based information mediation with MIX. *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. ACM Press.
- CAREY, M. J., PETKOVIC, D., THOMAS, J., WILLIAMS, J. H., WIMMERS, E. L., HAAS, L. M., SCHWARZ, P. M., ARYA, M., CODY, W. F., FAGIN, R., FLICKNER, M., LUNIEWSKI, A. W. & NIBLACK, W. (1995)

- Towards heterogeneous multimedia information systems: the Garlic approach. *RIDE '95: Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95)*. IEEE Computer Society.
- FUNDERBURK, J. E., KIERNAN., G., SHANMUGASUNDARAM, J., SHEKITA, E. & WEI, C. (2002) XTABLES: Bridging Relational Technology and XML. *IBM Systems Journal*, 41(4), 616-641.
- GARDARIN, G., SHA, F. & DANG-NGOC, T. (1999) XML-based Components for Federating Multiple Heterogeneous Data Sources. *ER '99: Proceedings of the 18th International Conference on Conceptual Modeling*. Springer-Verlag.
- KARUNARATNA, D. D., GRAY, W. A. & FIDDIAN, N. J. (May 1998) Organising Knowledge of a Federated Database System to Support Multiple View Generation. *Proceedings of the 5th KRDB Workshop (Knowledge Representation meets Data Bases)*, pp. 12.1-12.10, Seattle, Washington, USA.
- KURGAN, L., SWIERCZ, W. & CIOS, K. (2002) Semantic Mapping of XML Tags using Inductive Machine Learning. *Proceedings of the International Conference on Machine Learning and Applications - ICMLA '02*. Las Vegas, Nevada, USA.
- LEE, K., MIN, J. & PARK, K. (2002) A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources. *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 7*. IEEE Computer Society.
- LEHTI, P. & FANKHAUSER, P. (2004) XML data integration with OWL: Experiences & challenges. *Proceedings of the International Symposium on Applications and the Internet (SAINT 2004)*. Tokyo, Japan.
- LENZERINI, M. (2002) Data integration: a theoretical perspective. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems*. Madison, Wisconsin.
- NAM, Y.-K., GOGUEN, J. & WANG, G. (2003) A Metadata Tool for Retrieval from Heterogeneous Distributed XML Documents. *Proceedings of the International Conference on Computational Science, LNCS 2660, Springer, pp. 1020-1029*.
- PAPAKONSTANTINOY, Y., GARCIA-MOLINA, H. & ULLMAN, J. D. (1996) MedMaker: A Mediation System Based on Declarative Specifications. *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*. IEEE Computer Society.
- SEGEV, A. & CHATTERJEE, A. (December 1991) Data manipulation in heterogeneous databases. *Sigmod Record*, 20(4), 64-68.
- SHANMUGASUNDARAM, J., SHEKITA, E. J., BARR, R., CAREY, M. J., LINDSAY, B. G., PIRAHESH, H. & REINWALD, B. (September 2000) Efficiently Publishing Relational Data as XML Documents. *Proceedings of the 26th International Conference on Very Large Databases, (VLDB2000)*. Cairo, Egypt.
- ULLMAN, J. D. (1997) Information Integration Using Logical Views. *ICDT '97: Proceedings of the 6th International Conference on Database Theory*. Springer-Verlag.
- WIEDERHOLD, G. (March 1992) Mediators in the Architecture of Future Information System. *IEEE Computer*, 25(3), 38-49.
- YOUNG-KWANG, N., JOSEPH, G. & GUILIAN, W. (October 2002) A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. *Proceedings of the Confederated International Conferences DOA, CoopIS and ODBASE, Irvine CA, LNCS 2519, Springer, pp. 1332-1344*.