A Generic XML-(Relational : Object) Mapping Scheme (GXROMS)

Munib A. Qataishat*, Shereen W. Al Haj Badar* and Majid Basito*

Received on Aug. 10, 2005 Accepted for publication on Dec. 21, 2005

Abstract

XML (eXtensible Markup Language) has been playing a major role in Data Exchange through the Internet, due to its high ability of organizing, describing, and structuring data. Thus, XML has been widely used for exchanging Relational Database Management Systems (RDBMSs) and Object Oriented Database Management Systems (OODBMSs) contents. This paper introduces a scheme that can be used for systematic XML-RDBMS transformations as well as XML-OODBMS transformations. The proposed scheme (GXROMS) contains three components: a generic front-end (Scans XML constructs), a Common Intermediate XML Interface and a generic back-end (produces DBMS constructs). Our choice of implementation of the generic front-end and generic back-end interfaces called JAVA Document Object Model (JDOM). JDOM represents a hybrid of the two famous schemes Document Object Model (DOM) and Simple API for XML parsing (SAX), maximizing the benefits of both and minimizing their drawbacks. The main advantage in the proposed scheme is that it is a generic transformation scheme rather than application specific transformations provided by the majority of already existing XML-RDBMS or XML-OODBMS mapping schemes. In addition, GXROMS strengthens its mapping schemes as it is capable of dealing with XML schemas as well as XML semi or non structured documents.

Keywords: XML, XML schema mapping, XML transformation, XML-Relational mapping, XML-Object-Oriented mapping

Introduction

DBMSs have become the basic building blocks for any successful, well-established business. A wide variety of DBMSs have been developed since the 60's of the last century [1,2]. Staring from Network and Hierarchical Database architecture, and followed by the evolution of Relational and Enhanced Relational Databases systems which were built on the sound basis of set theory. The DBMSs continued to deploying Object-Oriented concepts into the world of Databases which resulted in the state-of-art DBMSs that are OODBMSs.

© 2006 by Yarmouk University, Irbid, Jordan.
* King Abdullah II School for Information Technology, University of Jordan, Amman - 11942, Jordan,
E-mail: maq@ju.edu.jo
Another area that has been witnessing an increased development through the last two decades is communication and networking technologies. The establishment of a reliable, fast and secure network infrastructure, including access to the Internet, has become a vital pre-request for any successful system.

The evolution of DBMS and the Internet has resulted in a still-hot challenge, data exchange through the Internet. Network clients all over the globe are always willing to be able to directly exchange files containing a mixture of texts, images, audio and video, regardless of heterogeneity of their underlying systems including DBMSs [3-6]. This challenge has resulted in the development of many languages that aimed at simplifying data exchange between online resources. Among these is XML (eXtensible Markup Language) [7-9] which is one of the most widely used data exchange language. The XML language has been created in 1998 and became a standard format for various data transfers [10-13]. The advantage of using such kind of transfer media brings some new features. Deploying XML for data exchange is a direct consequence of its flexibility and capability of representing different data formats, either structured or semi-structured [1, 12, 16, 17].

Many DBMS specific solutions have been developed so far [10, 11, 13, 14], in order to allow data to be exchanged via the web. However, these solutions lack generality in the sense that they are applicable for the specific DBMS for which they were originally developed. Thus, their reusability is definitely limited. Moreover, mapping schemes made to allow data exchanges among heterogeneous DBMSs do still need extensive elaboration.

This paper proposes a generic scheme for allowing data exchange among heterogeneous as well as homogeneous DBMSs (RDBMSs and OODBMSs). The proposed scheme is more generic than the model described in [4] in the sense that its applicable for any RDBMSs or OODBMSs syntax. Thus, it is not an application specific solution.

Section 2 highlights related work briefly. A detailed description of the proposed scheme is depicted in Section 3. In addition it provides comparative analysis of the proposed scheme against some of the already existing solutions. Section 4 concludes the work done in this paper.

Literature Overview

Several schemes that aim at achieving data exchange among homogeneous as well as heterogeneous environments have been developed so far. Some of these schemes are Automatic XML Interchange System (AXIS) [17] and Load/Extract Utility System (LEUS) [4]. AXIS is a data exchange architecture used to allow data interchange among heterogeneous enterprises using XML, whereas, LEUS system is limited to relational database transformation.

AXIS utilizes XML Document Type Definitions (XML DTDs) since XML has become a de facto standard data format. AXIS maps XML DTDs structures into database structures. These database structures are stored in AXIS Meta model. The Meta model is
A Generic XML-(Relational:Object) Mapping Scheme (GXROMS)

consulted whenever XML document generating the data given in the database and vice versa. The architecture of AXIS is depicted in [17].

AXIS consists of three components:

1. Relational Database meta data model.
2. XML meta data model.
3. Correspondence model.

Data exchange schemes between relational Databases using XML has been extensively investigated. For instance, in [16] it has been found that some researchers focus on offline data transfers. They also assume the existence of an XML interface as a part of the designated RDBMS. The mapping schemes provided in their work depend on the mapping described in [3]. They also concentrate on data validation for data incoming from XML documents into the RDBMS. Other mapping schemes focus on specific DB products such as Oracle and Sybase [1,6] mappings into XML and vice versa.

A well composed DTD-RDBMS mapping scheme is depicted in [3] by Bourret. He discusses table-based mapping as well as object-relational mapping among heterogeneous data sources. In Bourret work an extensive XML mapping into either RDBMS or OODBMS is described including: mapping DTDs to object schemes, mapping object schemes to database schemes, and mapping complex data models.

The mapping scheme proposed by Bourret considers mapping choices, repeated children, optional children, subgroups, and mixed content. Moreover, attribute mappings are highlighted as well as alternate mapping.

Bourret work also considers mapping W3C schemes to object schemes. However, he concluded that mapping XML schemes to object schemes is not a good idea if it can be avoided. This is due to the fact that mapping XML scheme to an object scheme might oblige applications to use data specific objects in order to process the data in the XML document transferred via the proposed scheme.

Three semantic based transformation algorithms are proposed by Dongwon Lee in [11] in order to convert XML documents to relational formats and vice versa. This scheme consists of:

1. CPI: Constraint Processing Inline algorithm. It converts XML scheme to a relational scheme while preserving semantic constraints of the original XML scheme.
2. NetT: Nested Translation algorithm. It derives a nested structure from a flat relational scheme by applying the net operator on the XML schema to make that schema Hierarchical.
A storage technique and mapping scheme for XML is described by Sihem in [18]. This work develops a new mapping scheme that identifies orthogonal aspects of mapping XML into relations. This work focuses on the support of storage techniques for mixed contents.

Generic XML-(RDBMS: OODBMS) Mapping Scheme(GXROMS)

Generic Mapping Scheme Design Goals

The design of mapping schemes is a difficult task, as it requires both general knowledge about database systems and detailed mapping schemes, and specific knowledge about customization process of the selected mapping schemes. Such types of knowledge require discussion of a mapping scenario.

In a heterogeneous environment, heterogeneous DBMSs cannot communicate directly via the web through schema integration process [19]. Thus, DBMS record fragments of existing data types (e.g. text, images, video, audio, ... etc) then these data types are classified and transferred into XML documents that can be either structured or semi-structured. At the recipient side these XML tags are transformed back into DBMS fragments. In order to rebuild the DBMS record fragments using received XML tags, many schemes have been developed so far. However, the majority of these mapping schemes are DBMS specific. That means that they are applicable for transformations among specific DBMSs like Oracle for instance [6]. This implies their limited reusability. The XML-(RDBMS: OODBMS) proposed mapping scheme overcomes their shortcoming by providing a generic mapping scheme that generates a generic RDBMS or OODBMS constructs as appropriate which can be easily converted into a specific RDBMS or OODBMS syntax.

Figure 3.1 depicts the architecture of GXROMS scheme. The figure describes the stages through which a given XML schema/document is transferred into the corresponding DBMS generic SQL constructs.

![Figure 3.1 The architecture of GXROMS system](image)

In current research, we have used XML schema style to define the allowable structure of elements for XML document rather than direct DTD, because XML schema is richer; provide an improved data typing system and having more advanced internal
structures than declarations in DTDs. Therefore, the above fornt-end interface through
Pre-XML processing supports conversion of DTDs to XML schemas, a more the
W3C's recommended XML data modeling language. Converting DTDs to XML schema
provides numerous powerful benefits, including:

- Support for primitive (built-in) data types (eg: xsd:integer, xsd:string, xsd:date, and
  so on), which facilitates using XML in conjunction with other typed-data, including
  relational data.

- The ability to define custom data types, using object-oriented data modeling
  principles: encapsulation, inheritance, and substitution.

The largest addition XML schemas provide to the functionality of the descriptions
is a vastly improved data typing system. XML schemas provide data-oriented data types
in addition to the more document-oriented data types XML 1.0 DTDs support, making
XML more suitable for data interchange applications. Built-in data types include strings,
booleans, and time values, and the XML schemas drafts provides a mechanism for
generating additional data types. Using this pre-processing, the draft provides support for
all of the XML 1.0 data types (NMTOKENS, IDREFS, etc.) as well as data-specific
types like decimal, integer, date, and time. Using XML schemas, developers can build
their own libraries of easily interchanged data types and use them inside schemas or
across multiple schemas.

The first, and probably most significant, difference between XML schemas and
XML DTDs is that XML schemas use XML document syntax. While transforming
the syntax to XML doesn't automatically improve the quality of the description, it does
make those descriptions far more extensible than they were in the original DTD syntax.
Declarations can have richer and more complex internal structures than declarations in
DTDs, and schema designers can take advantage of XML's containment hierarchies to
add extra information where appropriate - even sophisticated information like
documentation. There are a few other benefits from this approach. XML schemas can be
stored along with other XML documents in XML-oriented data stores, referenced, and
even styled, using commercial tools.

The stages through which XML schemas/documents are transferred into (RDBMS:
OODBMS) require processing XML schemas/documents by IXMLI (Common
Intermediate XML Interface). This generates the IXML (Intermediate XML)
file which is then examined against the syntax directed translation processor resulting in
a file that consists of generic SQL constructs that allow XML data fragments to be stored
consistently in any given RDBMS or OODBMS provided that a proper converter has
produced the specific DDL and DML statements corresponding to the designated
DBMS.

Typical XML documents can be classified into either DTD (Document Type
Definition)-based and non-DTD based documents. XML-DTD documents are highly,
regularly structured documents that adhere to certain structure described in the beginning
of the XML file. All subsequent tags in that file convey to the DTD rules. On the other
hand, non DTD-based XML documents have tags that do not obey some specific tag
structure. Consequently, these documents are not highly structured as opposed to the DTD-based ones. The preprocessing carried out by pre-CIXMLJ (Common Intermediate XML Interface) aims at extracting the DTD rules from DTD-based as well as non DTD-based documents. If the processed file is already a DTD-based document, then the DTD in the file can be extracted forwardly. However, non DTD-based XML files need careful consideration and heavy pre-processing in order to extract the DTD tags implied in them. This has been accomplished by introducing a new syntax and semantic transformation grammar which extract accumulative knowledge based on graph algorithms and intelligent DTD-XML knowledge extraction.

In this case, we have classified the transformations into two categories: structure-mapping approach and model-mapping approach [16]. In the former, the design of database schema is based on the understanding of DTD (Document Type Descriptor) that describes the structure of XML documents. In the latter a fixed database schema is used to store any XML documents without assistance of DTD. We have concluded that a combination of graph algorithms and model-mapping approach can solve many problems associated with non DTD based XML constructs. The major product of this work we called intelligent DTD-XML knowledge extractor. It has been found comparable to other approaches in terms of its capability in supporting DTD and non-DTD transformation as well as its capability of supporting any sophisticated XML applications that are considered either as static or dynamic. For space limitation and simplicity, detailed description of this approach will be the subject of another paper.

**Implementation Decision of the XML Parser**

For parsing XML files, two dominant models exist: DOM (Document Object Model) and SAX (Simple API for XML parsing) [20,21]. SAX is an event driven API that calls event methods whilst parsing the document, i.e. it provides methods that can react to data in an XML document at the time that data being read. This is simple and obvious in three cases: if we’re only interested in a few parts of processed document; if we know how to locate those parts within the stream of SAX events or if the sequence of data is important. DOM on the other hand, parses the entire document and creates a corresponding Document object that can be browsed using appropriate method calls. Since DOM creates the object from the XML file, it requires enough memory to hold the file. This may be a disadvantage if the file is very large and available memory is very small. In effect, DOM provides programmatic access to the entire document, in a non-linear order. Though the DOM model is easier to use, but the SAX model allows faster parsing and requires less memory.

Our choice of implementation is a modified form of DOM called JDOM. JDOM was selected because it attempts to provide a seamless XML parser and builder as API. It is a hybrid of SAX and DOM, maximizing the benefits of both and minimizing their drawbacks. Therefore, JDOM model allows faster parsing and requires less memory than DOM in addition it is easier to use than SAX because it provides programmatic access to the entire document, in a non-linear order.
Mapping Model Components of Common Intermediate XML Interface (C1XML-I)

Tags can be used freely in an XML document or can be used in accordance with the document type definitions (DTDs) or XML Schema which define the types for a class of documents. An XML document that conforms to a DTD or XML Schema is called a valid XML document. A DTD or XML Schema is used to define the allowable structure of elements in a valid XML document. A DTD can include four kinds of declarations: element type, attribute-list, notation and entity. An element type declaration is analogous to a data type definition; it names an element and defines the allowable content and structure. An element may contain only other elements (called element content) or may contain any mix of other elements and text, which is represented as PCDATA (called mixed content). An EMPTY element type declaration is used to name an element type without content. Finally an element type can be declared with content ANY meaning the type of the element is arbitrary. Attribute-list declarations define the attributes of an element type. The declaration includes attribute names, default values and types.

XML Schema, which is also used to define the allowable structure of elements for a given application or application domain in a valid XML document, uses XML document syntax. Declarations in XML Schema can have richer and more complex internal structures than declarations in DTDs. Schema designers can take advantage of XML's containment hierarchies to add extra information where appropriate. XML Schemas also provide an improved data typing system. They provide data-oriented data types in addition to the more document-oriented data types that XML 1.0 DTDs support, making XML more suitable for data interchange applications. Built-in data types include strings, booleans, and time values. The XML Schema draft provides a mechanism for generating additional data types. Besides, XML Schema supports namespaces and the notion of keys to uniquely identify elements in an XML document.

Several mapping rules injected in C1XML-I mapping model to perform various XML mapping concepts. There is an obvious mapping between some XML elements and their correspondence data modeling concepts of RDBMS. But in case of XML-OODBMS mapping, it needs extra attention because of advanced concepts such as class hierarchy, association, aggregation, etc...[1,2]. Therefore, in this section we will concentrate on some OO mapping scheme.

In general our system can handle transformations shown in table 1.

<table>
<thead>
<tr>
<th>XML concept</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping - XML elements</td>
<td>Complex and Simple</td>
</tr>
<tr>
<td>Mapping - XML attributes</td>
<td>All</td>
</tr>
<tr>
<td>Mapping - Relationships</td>
<td>Association (1:1 , 1:n )</td>
</tr>
<tr>
<td>Mapping - XML data</td>
<td>All</td>
</tr>
</tbody>
</table>
XML elements shown in table 2 can be complex or simple elements. Complex element can have children elements, while simple elements have no children elements. If XML element is a complex element, then the element is mapped to new class, if it has a parent element, then it is mapped to a reference attribute (pointer to object) for the current class in the parent class. If the XML element is simple then it will be directly mapped to attribute in the parent element (class).

The following table shows some important elements used to define XML schema.

<table>
<thead>
<tr>
<th>Element</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation</td>
<td>Specifies the top-level element for schema comments</td>
</tr>
<tr>
<td>attribute</td>
<td>Defines an attribute</td>
</tr>
<tr>
<td>complex type</td>
<td>Defines a complex type element</td>
</tr>
<tr>
<td>documentation</td>
<td>Defines text comments in a schema (must go inside annotation)</td>
</tr>
<tr>
<td>element</td>
<td>Defines an element</td>
</tr>
<tr>
<td>notation</td>
<td>Describes the format of non-XML data within an XML document</td>
</tr>
<tr>
<td>schema</td>
<td>Defines the root element of a schema</td>
</tr>
<tr>
<td>sequence</td>
<td>Specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times</td>
</tr>
</tbody>
</table>

For simplicity, the following algorithm shows abstract mapping of (XML schema elements to OO data modeling concepts):

Algorithm: MapTree_Class_Hierarchy
Input: XML Root.Class_Hierarchy element
Start: While <> end of Root.Class_Hierarchy children

2. Begin

3. Element e = Root.Class_Hierarchy child
4. If Root.Class_Hierarchy child is complex Element then
   5. Map e to class e
   6. Map e.AttributeChildren to Attributes in class e with same type
   7. If e.Parent is element then
      8. Map Parent as attribute in class e with reference type to parent
   9. End If
10. List Elements d = e.getChildren()
11. i = 1
12. While <> end of d elements
13. Begin
14. If element d[i] is simple element then
15.   Map element d[i] as Attribute in class e with same type
16. Else if element d[i] is complex element then
17.   If d[i].getAttribute(maxOccurs) = unbounded then
18.      Association type = One to Many
19.   Map d[i] to attribute in class e with type as list of reference to d[i] object
20.   Else if d[i].getAttribute(maxOccurs) <> 1 then
21.      Association type = One to One

342
A Generic XML-(Relational : Object) Mapping Scheme (GXROMS)

23. Map d[j] to attribute in class e with type as
24. reference to d[i] object
25. End if
26. End if
27. MapTrue_Class_Hierarchy(d[j])
28. End if
29. End if
30. Else
31. Error Message
32. End if
33. End While
34. End While

Syntax and Semantic Based Mapping

Each of the elements or attributes being processed will be associated with two attributes: type_t and null_t. type_t can have the value of: SIMPLE, COMPLEX or ATTRIB. ATTRIB stands for XML defined attributes, whereas SIMPLE stands for atomic XML elements, and COMPLEX stands for elements that consist of other elements (i.e., nested elements). This classification will play a vital role in the subsequent mapping. The second attribute nullable_t is a Boolean attribute which can be either true or false. If the existence of the element is necessary (i.e. non nullable) then the value will be false, whereas if the existence of the element is unnecessary, then the value of nullable_t will be true. Another attributes associated with each elements are declare and sequence which are going to be utilized by the syntax directed translation scheme.

The next step is the CIXMLI processing. In this step the extracted DTD and data tags from the previous step are examined in order to:

1. Eliminate wildcards like: *, +, ?, . , etc and replace them with sound interpreting XML tags. In addition to setting the value of the nullable_t for every element by either true, false or empty. Entities that are found to be attributes (t_type = ATTRIB) will have their nullable attribute to be false if the Keyword #REQUIRED appears in their definition and will have it true otherwise.

2. Secondly, classify all of the DTD parsed elements into either simple or complex, by specifying the value of type_t attribute for each.

Tags declaring elements with wild cards can not be directly mapped into any SQL statement. Therefore, they need intermediate processing and transformation. The following set of examples describes how these wild cards can be omitted.

Example 1:

<ELEMENT chapter(title, section+)>  

+ wildcard means that the associated element can appear once or more times in any subsequent chapter tag. Such an element definition will be replaced with the following definition:

<ELEMENT chapter(title, section_1, section_2,...,section_n)>
The value of (n) can be determined from the currently parsed XML document, by counting the maximum number of sections in all chapter data tags. The first section (i.e., section_1) will have its nullable_t true, whereas the remaining sections will have it false. The title element will have its nullable_t true. The nullable_t value for chapter will be empty if chapter is not appearing as a component within another element in that XML document.

Example 2:

```xml
<ELEMENT chapter(title, section*)>

*wildcard means that the associated element can appear zero or more times in any subsequent chapter tag. Such an element definition will be replaced with the following definition:

```xml
<ELEMENT chapter(title, section_1, section_2,...,section_n)>

the value of (n) will be determined as in Example 1. The assignment of the values of the nullable_t attribute is also as in Example 1 with one exception: section_1.nullable_t will be false.

Example 3:

```xml
<ELEMENT chapter(title, section?)>

? wildcard means that the associated element can appear zero or once in any subsequent chapter tag. Such an element definition will be replaced with the following definition:

```xml
<ELEMENT chapter(title, section_1)>

here its clear that title.nullable_t will be false, whereas section_1.nullable_t will be true. The value of chapter.nullable_t will be determined as depicted in Example 1.

Another symbol that needs being eliminated is \|: or connecter.

Example 4:

```xml
<ELEMENT chapter(title|section)>

\| operator means that in any chapter tag, either one section component or one title component will appear and not both. Such an element definition will be replaced with the following two definitions:

```xml
<ELEMENT chapter(title)>

```xml
<ELEMENT chapter(section)>

here its clear that title.nullable_t will be false, as well as section.nullable_t which will be also false. The value of chapter.nullable_t will be determined as depicted in Example 1.

However, in many cases the structure of declared elements is not as easy as depicted in the above examples. It usually consists of a mixture of the different
operators. Thus, the following syntax directed grammar describes the generation of all possible valid element definitions using the operators "\( {}^* \), "\( {}^+ \), "\( {}^- \), "\( {}^? \), "\( {}^+ \), "\( {}^* \)."

1. \( E::= (E) \quad E\text{.copies} = E_1\text{.copies} \)
2. \( E::= E_1E \quad E\text{.copies} = 1, E_1\text{.copies} = 1 \)
3. \( E::= E_1^* | E_1^+ \quad E\text{.copies} = n \)
4. \( E::= E_1^? \quad E\text{.copies} = 1 \)
5. \( E::= E_1[E \quad (E \text{ is either } E \text{ or } E_1) \text{ split the definition into two definitions on using } E \text{ and the other using } E_1. \text{ Moreover, let } E_1\text{.copies} = E\text{.copies and } E_2\text{.copies} = E\text{.copies} \)
6. \( E_1::= E \quad E\text{.copies} = E_1\text{.copies} \)
7. \( E_1::= id \quad id\text{.copies} = E_1\text{.copies} \)
8. \( id::= \text{ the set of elements names deduced by the pre-CIXML generic front end.} \)

After applying the above rules to any element definition, the final result will be associating each id in that definition, and newly generated ones if any, with a proper number of copies. The value of \( (n) \) will determined as previously discussed. Then, that id will be replaced in the definition in which it appears by the designated number of replicas. For instance, if the element A ,which appeared in the definition of some element B had the number of copies 4, then the occurrence of A will be replaced with: A_1, A_2, A_3, A_4.

The second task accomplished by CIXML interface is classifying each of the elements deduced from pre-CIXML front end into either: SIMPLE, COMPLEX , or ATTRIB. In order to do so, the CIXML interface parses the XML document received, checking each of the entities found in the lists deduced by the front end preprocessor. If a given entity, call it T, appears in an attribute tag, then T. type = ATTRIB, whereas if it never appears within the definition of any other element, then T. type = COMPLEX. Remaining elements will be assigned type = SIMPLE.

The processing done to the XML file by CIXML interface results in what we will call CIXML document. This document will be passed to the next stage of processing, the syntax directed translation processing shown in figure 3.1. This stage aims at generating corresponding generic DDL and DML from the designated CIXML document. The following set of syntax directed translation rules with their associated actions are used to accomplish this task.

1. \( mode ::= \#REQUIRED | \#DEFAULT \)
2. \( type ::= \text{ a type defined in XML documents} \)
3. \( ATT(A)::= <ATTLIST A t \text{ type mode}> \rightarrow Att(A).\text{declare} = t \text{ RDBMS.cast(type)} \)
4. \( simple ::= <ELEMENT s \text{ type}> \rightarrow \text{simple.declare} = s \text{ RDBMS.cast(type)} \)

345
5. \( \text{simple} := \langle \text{ELEMENT} \ s\ \text{type}> \ \text{Att}(s) \ \rightarrow \ \text{simple.declare} = s \ \text{RDBMS.cas}(\text{type}) \ || \ \text{"",} \ || \ \text{Att}(s).\text{declare} \)

6. \( \text{simple} := \langle \text{ELEMENT} \ s\ \text{type}> \ \text{Att}(s)_{1}, \text{Att}(s)_{2}, \ldots, \text{Att}(s)_{n} \ \rightarrow \ \text{simple.declare} = s \ \text{RDBMS.cas}(\text{type}) \ || \ \text{"",} \ || \ \text{Att}(s)_{1}.\text{declare} \ || \ \text{Att}(s)_{2}.\text{declare} \ || \ \ldots \ || \ \text{Att}(s)_{n}.\text{declare} \)

7. \( \text{complex} := \langle \text{ELEMENT} \ \text{th}_\text{name}(\text{simple}_{1}, \text{simple}_{2}, \ldots, \text{simple}_{n}) \rangle \ \rightarrow \ \text{complex.declare} = \)

\[
\begin{align*}
\text{create table } &\text{th}_\text{name}(\text{simple}_{1}.\text{declare} \ || \ \text{"",} \ || \ \text{simple}_{2}.\text{declare} \ || \ \ldots \ || \ \text{simple}_{n}.\text{declare} \ || \ \text{"",} \ || \\
\text{primary key } &\text{th}_\text{name}.\text{sequencer} \ || \ \text{"",} \ || \\
\text{foreign key null} &) \\
\text{complex.ref} = &\text{th}_\text{name}.\text{sequencer} \\
\text{append} \ \text{complex.declare} &\text{to generic RDBMS file}
\end{align*}
\]

9. \( \text{complex} := \langle \text{ELEMENT} \ \text{th}_\text{name} (\text{simple}_{1}, \ldots, \text{simple}_{n}, \text{complex}_{1}, \ldots, \text{complex}_{m}) \rangle \ \rightarrow \ \text{complex.declare} = \)

\[
\begin{align*}
\text{create table } &\text{th}_\text{name}(\text{simple}_{1}.\text{declare} \ || \ \text{"",} \ || \\
\ldots \ || \ &\text{simple}_{n}.\text{declare} \ || \ \text{"",} \ || \\
\text{primary key } &\text{th}_\text{name}.\text{sequencer} \ || \ \text{"",} \ || \\
\text{foreign key } &\text{key}_{1} = \text{complex}_{1}.\text{ref} \ || \ \text{"",} \ || \\
\text{foreign key } &\text{key}_{2} = \text{complex}_{2}.\text{ref} \ || \ \text{"",} \ || \\
&\ldots \\
\text{foreign key } &\text{key}_{m} = \text{complex}_{m}.\text{ref} \ || \ \text{"",} \ || \\
\text{complex.ref} = &\text{th}_\text{name}.\text{sequencer} \\
\text{append} \ \text{complex.declare} &\text{to generic RDBMS file}
\end{align*}
\]

10. \( \text{complex} := \langle \text{ELEMENT} \ \text{th}_\text{name} (\text{complex}_{1}, \ldots, \text{complex}_{m}) \rangle \ \rightarrow \ \text{complex.declare} = \)

\[
\begin{align*}
\text{create table } &\text{th}_\text{name}(
\end{align*}
\]

346
A Generic XML-(Relational : Object) Mapping Scheme (GXROMS)

\[ \text{primary key th\_name, sequencer} \]
\[ \text{foreign key}_1 = \text{complex}_1, \text{ref} \]
\[ \text{foreign key}_2 = \text{complex}_2, \text{ref} \]
\[ \ldots \]
\[ \text{foreign key}_n = \text{complex}_n, \text{ref} \]
\[ \text{complex}\_\text{ref} = \text{th\_name, sequencer} \]
\[ \text{append} \quad \text{complex, declare to generic RDBMS file} \]

Comparison of GXROMS with Existing Mapping Schemes

This section aims at comparing the GXROMS with already existing XML-DBMSs mapping schemes in terms of transformation accuracy, transformation quality, customization (i.e. generic mapping) and automation factors rather than time and space complexity factors. The mapping schemes proposed in [3-5, 18] ignore the semantics of the scheme being mapped. This implies that XML-DBMS mappings will result in inaccurate interpretations due to the propagation of semantic ignorance between XML-RDBMS and RDBMS-OODBMS mappings.

Another problem associated with the schemes proposed by Bourret [3] is the that they did not provide a systematic technique to resolve complex content models (i.e. XML tags with wildcards (+, ?, * , .). These problems do not exist in GXROMS since it is dynamic grammar based mapping technique rather than a static mapping that is both syntactic and semantic driven.

The AXIS scheme proposed in [19] requires the association between the RDBMS model and the XML model to be manual rather than automatic. This implies that the quality of the mapping scenario will vary from a user to another. Moreover, such a mapping scheme imposes an additional overhead due to users' necessary intervention. Such manual association is not required in GXROMS.

The mapping solution suggested in [6, 14] is specific for some special RDBMS that provide support for parsing XML schemes and converting them to the corresponding DDLs and DMLs. This implies that the mappings in [14] lack generality, whereas GXROMS is generic and does not depend on any certain underlying DBMS. It also does not expect the existence of any parsing support for XML various documents.

The mapping technique suggested in [18] focused on mapping XML-RDBMS using grammar rules written in XML itself. However, some of the depicted mapper components rely on user provided data which might affect the quality of the mapping obtained from one user to another. Moreover, the mapping schemes considered XML schemes and DTDs, but no clear processing was provided for XML semi-structures or non-structured documents.
Conclusions

Several mapping schemes suggested in the literature for the XML-RDBMS transformations as well as XML-OO DBMS transformations. None of these schemes have tackled the problem of customization and generic mapping. Our research proposed design and development of a scheme that can be used for systematic XML-RDBMS transformations as well as XML-OO DBMS transformations. Main design goal of the system is to produce generic transformation scheme rather than application specific transformation provided by the majority of already existing XML-RDBMS or XML-OO DBMS mapping schemes. This has been achieved due to the powerful architectural design of the system (GXROMS) which has strengthened its functionality. At the heart of the system, several mapping rules injected in the Common Intermediate XML mapping model to perform various XML mapping concepts. It has been found that an obvious mapping between some XML elements and their correspondence data modeling concepts of relational database concepts. But in case of Object-Oriented concepts mapping, case based simulation modeling proposed to cope with some advanced concepts of Object-Oriented data modeling such as class hierarchy, association, and aggregation. In addition, many heuristic mapping rules combined within mapping algorithms developed on JAVA language using JAVA Document Object Model (JDOM) API to parse the XML schema. Our choice of implementation of the two interfaces (generic front-end and generic back-end interfaces) outperformed their counterparts SAX and DOM schemes by maximizing the benefits of both and minimizing their drawbacks. The proposed scheme have been tested successfully and proved to be highly customizable.

الخطط نموذج تحويل عام بين لغة XML و قواعد البيانات المتصلة والكنينية

منيب قطريشات، شريف العجم بدرا، محمد سبيسي

ملخص

تتطلب لغة XML (eXtensible Markup Language) دوراً رئيسيًّا في تبادل المعلومات عبر الإنترنت. ويجري ذلك بالاعتماد على نظم وصف وهيكلة البيانات، لذلك تم استخدامها على نطاق واسع في تطبيقات تبادل المعلومات عبر قواعد البيانات المتصلة والكنينية. تقدم هذه الورقة البحثية نموذج يمكن استخدامه بطريقة منتظمة لتضليل البيانات وتحويل بين لغة Relational Database XML ومتطلبات XML و كذلك تبادل البيانات وتحويل بين لغة ومتطلبات هاكل Management Systems (RDBMS) و Object-Oriented Database Management XML إلى ما يناسبهما في قواعد البيانات الكينينية XML والنموذج المقترح على ثلاث مكونات رئيسية: واجهة التخطيط الأساسية.
A Generic XML-(Relational: Object) Mapping Scheme (GXROMS)

References


