ISSN 1746-7659, England, UK
Journal of Information and Computing Science
Vol. 1, No. 5, 2006, *pp*. 266-274

XML-RDB Driven Semi-Structure Data Management

Jingtao Zhou [+], Shusheng Zhang, Mingwei Wang, and Hongwei Sun

The Key Laboratory of Contemporary Design and Integrated Manufacturing Technology, Ministry of Education, Northwestern Polytechnical University, Xi'an, China, 710072

(*Received February 06, 2006, accepted May 03, 2006*)

Abstract. Nowadays, swirling around most businesses is not structure information but semi-structure information, such as spreadsheets, and documents created by an individual or company. However, in spite of success of data management of structure data by using databases, current enterprise information infrastructure is poorly suited for dealing with the continuing, rapid explosion of semi-structure data in enterprise. In this paper we discuss a system based on the integration between XML and RDB in a three-tiered framework, which undertakes research at the intersection of XML and RDB, exploits their strengths in a common framework, and expands their applicability in the area of semi-structure data management. Through a discussion of the fundamental architecture in general and its components in particular, we depict the basic principles that should form the basis for the design of the architecture based on the integration of XML and RDB. The key contribution of this paper is the framework for semi-structure data management, which is orchestrated through the transformation of representation, query and updating between XML data and RDB data.

Keywords: Semi-structure Data, XML, RDB, Bi-directional Integration

1. Introduction

More and more data in enterprise today is semi-structure data rather than structure data. However, in spite of extensive R&D and successful pilots, a serious barrier to improvement exists: a severe lack of infrastructure for effective management and reuse of this semi-structure data in an ever-growing and dynamic business environment. One underlying problem has remained unsolved: semi-structure data resides in a great deal incompatible formats and cannot be systematically represented, managed, queried, integrated, unified or cleansed.

Meanwhile, XML is expected to be an ideal representation for not only structure data but also semi-structured data [1] since it can encode all kinds of data that is exchanged between computers, using XML Schemas to prescribe the data structure [2]. XML has been being widely adopted as a standard for information representation in enterprise domain. However, in spite of extensive R&D and successful pilots, XML technologies fall short where an enterprise needs to manage and use semi-structure data represented by XML just like database for structure data. This is the common occasion in an enterprise. In this context, we propose a new technology based on the bi-directional integration between XML and RDB to enables the mutual compensation of these two technologies. Storing semi-structure data in XML format into RDB can fully take advantage of the mature RDB technology; publishing RDB data into corresponding XML format enables semi-structure data to represent exchange and use standardly.

The remainder of this paper is organized as follows. In Section 2, we compare XML and RDB from technology aspect. In section 3, we introduced the technology of the Three-tiered Bi-directional Integration between XML and RDB. In section 4, we introduce the realization of the technology. Finally some result test, concluding remarks and future works are given in section 5.

2. XML vs. RDB

Before we start talking about the integration between XML and RDB, we need to discuss the relation between them. An XML document is not a database only in the strictest sense of the term. That is, it is a

[+] *Tel.*: +86-29-8849 3232 ext.322; *fax*: +86-29-8849 1576; *E-mail address*: zhoujt@mail.nwpu.edu.cn

collection of data. In many ways, this makes it no different from any other file, which contains data of some sort. As a "database" format, XML has some advantages. For example, it is self-describing, which means the markup describes the structure and type names of the data, but not the semantics, it is portable and Unicode, and it can describe data in tree or graph structures. It also has some disadvantages. For example, it is verbose and access to the data is slow due to parsing and text conversion.

XML and its surrounding technologies constitute a "database" in the looser sense of the term that is, a database management system (DBMS). On the plus side, XML provides many of the things found in databases: storage (XML documents), schemas (DTDs, XML schema languages), query languages (XQuery, XPath, XQL, XML-QL, QUILT, etc.), programming interfaces (SAX, DOM, JDOM), and so on. On the minus side, it lacks many of the things found in real databases: efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on.

Thus, while it may be possible to use an XML document or documents as a database in environments with small amounts of data, few users, and modest performance requirements, this will fail in most environments, which have many users, strict data integrity requirements, and the need for good performance. The only real advantage of XML is that the data is portable, and this is less of an advantage than it seems due to the widespread availability of tools for serializing databases as XML.

3. Integration between XML and RDB

From the above comparison, we can see that RDB is more advanced than file system in bottom data management, such as efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, while in the data manipulation field, especially in information sharing and data exchange field, XML has much more advantage such as self-describing, portable and Unicode. Therefore, we integrate the advantages of the two technologies to offer a key support for the semi-structure data management and usage. Our integration is in three tiers: firstly, XML data is transformed into RDB and RDB data published into XML format based on schema mapping, which is called the first tier, i.e. Static Transformation Tier in the three-tiered structure. After Static Transformation between them, we consider data is stored in bottom RDB and it's XML view is used to exchanging information between applications. Then on the basis of Static Transformation Tier, we transform XML Query from user interface into SQL on the corresponding bottom RDB and return back the query result in XML format, which is the second tier named Dynamic Query Tier. At last, synchronous updating for the virtual XML and bottom RDB is carried out as the third tier named Synchronous Updating Tier.

3.1. Static transformation tier

The logical data structure is described by XML-Schema and relational schema respectively in XML and RDB, so the key point in transforming XML into RDB and RDB into XML is to establish an effective mapping algorithm between them. A complete data model includes data structure, data manipulation and data constraints. Hereinto data manipulation is exhibited in dynamic query and synchronous updating tiered, and the right data structure as well as the complete data constraints is ensured in the transformation algorithm. However, the exiting transformation algorithms focus only on the data structure but ignore data constrains, and they all adopt DTD as the schema description for XML. In our previous work, we have proposed schema-mapping based transformation algorithms between XML-Schema and relational schema [3] [4], which can ensure the integrated constraints mapping on the basis of accurate data structure mapping.

3.1.1. Transformation from XML to RDB

The transformation from XML to RDB is shown in figure 1. We map XML Schema to relational schema, and then translate data in XML instance document into relational database as tuples. Our main idea of the mapping algorithm from XML-Schema to relational schema is as follow. Firstly, a formal representation method named FD-XML is proposed for XML-Schema based on the regular tree grammar. FD-XML can integrally represent the information of XML-Schema in both the data structure and data constraint. At the same time, we put forward the extended ER model, EER, which includes two parts: diagram and accessories.

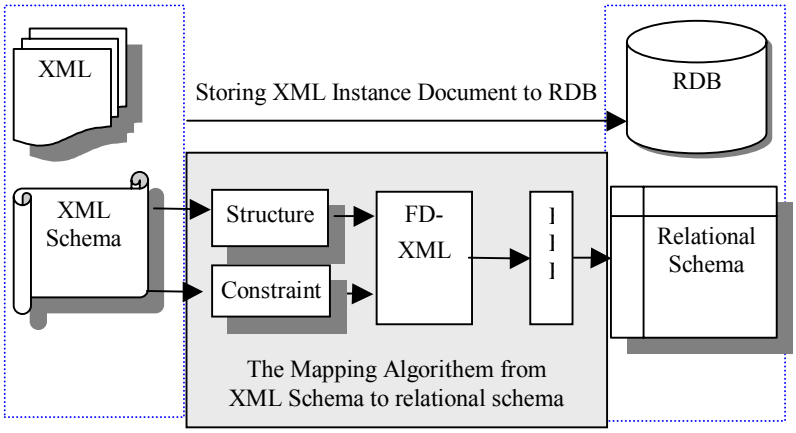

Fig. 1: Storing XML instance document to RDB.

3.1.1.1. FD-XML

We assume the existence of a set $\breve{N}$ of non-terminal names, a set $\breve{T}$ of terminal names and a set $\breve{A}$ of atomic data types defined in [5]. We also use the notations: ε denotes the empty string, "+" for union, "," for the concatenation, "$a^?$" for zero or one occurrence, "a^*" for the Kleen closure, and "a^+" for the "a, a^*". Now a RTG (regular tree grammar) based formalization expression for XML-Schema, FD-XML can be given.

An FD is denoted by an 8-tuple $FE = (N, T, S, E, A, C, U, K)$, where,

1) N is a set of non-terminal symbols, where $N \subseteq \breve{N}$;

2) T is a set of terminal symbols, where $T \subseteq \breve{T}$;

3) S is a set of start symbols, where $S \subseteq \breve{N}$;

4) E is a set of element production rules of the form "$X \to aRE$", where $X \in N$, $a \in T$, and RE is the content model of this production rule; it is:

$$RE ::= \varepsilon \,|\, n \,|\, \tau \,|\, (RE) \,|\, RE + RE \,|\, RE, RE \,|\, RE? \,|\, RE^* \,|\, RE+,$$

where $N \in \breve{N}$, $\tau \in \breve{A}$.

5) A is a set of attribute production rules of the form "$X \to aRE$", where $X \in N$, $a \in T$, and $RE ::= \varepsilon \,|\, a \,|\, (RE) \,|\, RE, RE$, where α is an attribute;

6) C is a set of terminal symbol data types and their constraints, $\forall c_i \in C$, c_i is an 18-tuple, $C_i(B_i, L_i, L_{\max i}, L_{\min i}, P_i, E_i, WS_i, MAX_{Ii}, MIN_{Ii}, MAX_{Ei}, MIN_{Ei}, TD_i, FD_i, Fix_i, Def_i, Opt_i, Pro_i, Req_i)$, where B_i denotes the basic user-defined data type of C_i, meanwhile, L_i for the *length* constraint, $L_{\max i}$ for the *maximal length* constraint, $L_{\max i}$ for the *minimum length* constraint, P_i for the *character pattern*, E_i for the *enumerate set*, WS_i for the *whiteSpace* constraint, MAX_{Ii} for the *maximal value* including itself, MIN_{Ii} for the *minimum value* including itself, MAX_{Ei} for the *maximal value* not including itself, MIN_{Ei} for the *minimum value* not including itself, TD_i for the *totalDigits* attribute, FD_i for the *fractionDigits* attribute, Fix_i for the *Fixed* attribute, Def_i for the *Default* attribute, Opt_i for the *Optional* attribute, Pro_i for the *prohibited* attribute and Req_i for the *Required* attribute;

7) U is a set of primary keys and unique constraints, $\forall pk_i \in U$, pk_i is a 3-tuple, $pk_i(K_{mi}, X_{Psi}, X_{Pfi})$, where K_{mi} denotes the name of pk_i, X_{Psi} denotes the *selector* domain expressed by Xpath of pk_i, and X_{Pfi} for the *field* domain expressed by Xpath of pk_i.

8) K is a set of foreign keys, $\forall fk_i \in K, fk_i$ is an 4-tuple $fk_i\left(K_{fi}, K_{mi}, X_{Psi}, X_{Pfi}\right)$, where K_{fi} denotes name of fk_i, K_{mi} denotes the referenced primary key in U, and X_{Psi} denotes the *selector* domain expressed by Xparth of fk_i, and X_{Pfi} denotes the *field* domain expressed by Xpath of fk_i.

3.1.1.2. Extend ER model to EER

The extension of E-R model is carried out in three aspects (see figure 2). Firstly, although the relationships represented by E-R model are abundant, there does not exit clear parent-children relationship, which is the main relationship in XML-Schema. We use the arrowhead starting from the parent element and ending at the sub-element to express the parent and subordinate elements in the parent-children relationship; (n, m) is then used to represent the occurrence time in XML-Schema, where n means the minimum occurrence time and m means the maximal. At last, as E-R model cannot express data semantic constraints perfectly while XML-Schema holds a powerful data constraints representing mechanism, which is to be represented in FD-XML by some sets, accessories are given to EER in order to preserve the integrality of data constraints.

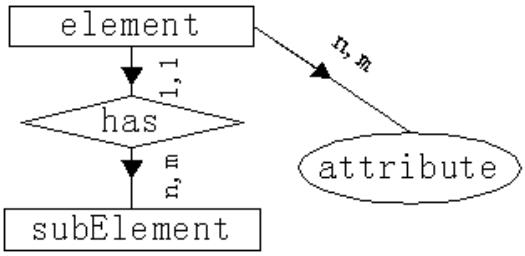

Fig. 2: Basic structure of EER diagram.

3.1.1.3. The transformation

FD-XML is then converted to EER model whose diagram represents the data structure and accessories contain the integrated data constraints, which is the formalization representation of XML-Schema, to EER. The detailed procedure is as follows:

1) Represent every element in the tuple N of FD-XML as an entity;

2) Build up the relationships between entities according to the tuple E of FD-XML, hereinto the relationships are denoted by diamond and the "father-son" relationship by arrowhead that start from the father element and end at the son element.

3) Build up the entities' attributes and confirm their occurrence times according to the tuple A of FD-XML.

4) Make the tuple C, U and K in FD-XML as the EER accessories.

Besides, to avoid the "data fraction" problem occurring in the converting process, we adopt the equivalence transformation method of graph theory in simplifying the EER diagram by reducing entities. Then, the simplified EER model is converted to the general relational schema, in this step, the normal form theory is introduced to optimize the general relational data model with correct and logical data structure, also the data constraints in the EER accessories is converted to the data constraints of relational schema to realize the non-loss conversion.

We refer reader to our previous work [3] for further discussion about this subject.

3.1.2. Transformation from RDB to XML

Similarly, our algorithm for publishing RDB to XML takes XML-Schema as its transformation goal and keeps the original entity integrality and reference integrality.

Since a data source is not likely to conform to XML Schema data model and data format, to preserve the data source's structures, constrains and so on, the translation must be performed based on schemas. The transformation process is composed of four steps as follows, shown in figure 3:

1) *Get Relational Metadata*. In this step the approach get metadata of relational schema through JDBC API.

2) *KRR*. Because the relational schema is flat but the XML Schema is hierarchical, to facilitate the

transformation we first use directed graph structure to represent the relational schema by an algorithm *KRR* (Key-based Relational Schema Reverse Reconstruction Arithmetic) in this step, and then translate the graph into XML hierarchical structure in next step. In order to preserve the necessary data types and constraints we define relational schema as a vector *R* and a corresponding relational directed graph as a vector *G* below.

Before definition, we assume that there exist a set $\breve{T}$ of table names, a set $\breve{C}$ of column names and a set $\breve{A}$ of atomic base types. When the column name conflict occurs, we qualify a column name by a table name with the notion ".", for example, *t.c*, $c\in\breve{C}$, $t\in\breve{T}$.

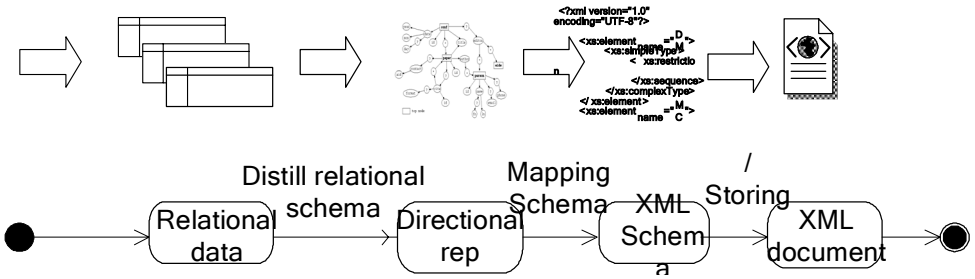

Fig 3: Publishing RDB data as XML format.

Definition 1. (Relational Schema) A relational schema is defined to be $R=(T,C,F,PK,FK)$, where

1) T is a finite set of table names in $\breve{T}$;

2) C is a mapping from a table name $t\in\breve{T}$ to a set of column names $c\in\breve{C}$;

3) F is a function that maps each column name $c\in\breve{C}$ to its column type definition: let α be the result of $F(c)$, where α is a 4-tuple (τ,μ,λ,n), where $\tau\in\breve{A}$, μ denotes whether a domain value of c can be unique, let $\mu=$"unique"or"*u* unique", λ denotes wxhether a domain value of c can be empty, $\lambda=$"nullable"or"*u* nullable", n is a finite set of some else type definition of c, denoted by ε;

4) PK is a finite set of key constrains: i.e., $\{c_i\}\xrightarrow{pkey}t$, and 5) FK is a finite set of referential constraints: i.e. $\{c_i\}\subseteq\{c_j\}$. Here, definition 1 is based on the Relational Schema definition according to [6].

Definition 2. (Relational Directed Graph) G is a pair $\langle V,E\rangle$, where V is a binary set of node, E is a set of edge. $\forall v_i(x,y)\in V$, where x denotes node v_i's type and y is the name or value of v_i. x and y are defined as table 1.

Table 1: Definition of *x* and *y* (according to Relational Schema).

x	Meanings of v_i	Values of y
x=1	Table node	t, where $t\in\breve{T}$
x=2	Column node	c, where $c\in\breve{C}$
x=3	Primary key node	pk, $pk\in PK$
x=4	Foreign key node	fk, $fk\in FK$
x=5	Atomic data types (τ node)	τ, $\tau\in\breve{A}$
x=6	μ node	unique or u_unique
x=7	λ node	Nullable or u_nullable

Table 1 also shows the mapping relationship between *Relational Schema* definition and *Relational Directed Graph* definition.

The algorithm is described as follows:

i. For each table $t\in T$ in R, create its corresponding table node t'.

ii. For each $c \in C(t)$ in R, create its corresponding column node c' and edge which is incident from the table node t' and incident to the column node c'.

iii. For each $\tau, \mu, \lambda \in a$, where $a \in F(c)$ in R, create their corresponding nodes τ', μ' and λ' and edges $e(c', \tau')$, $e(c', \mu')$ and $e(c', \lambda')$, where $e \in E$.

iv. For each $pk = \{\{c_i\} \xrightarrow{pkey} t\} \in PK$, create its corresponding primary key node pk' and edges $e(c_i', pk')$, where $e \in E, c_i'$ is the corresponding node of c_i.

v. For each $fk = \{\{c_i\} \subseteq \{c_j\}\} \in FK$, create its corresponding foreign key node fk' and edges e (c_i', pk'), $e(pk', c_j')$, where $e \in E, c_i'$ and c_j' are the corresponding nodes of c_i and c_j'.

3) *CPM*. In this step we translate the previous step's results into the XML Schema representations by *CPM* (Constraints-Preserving Mapping from Relational Schema to XML Schema) algorithm, which defines a schema mapping template through the segmental modeling of XML Schema and transform the relational schema described using directed graph to XML Schema representations according to the schema mapping template.

The mapping template is mainly composed of XML Schema's *complexType* entities and constraints components including *key*, *keyref* and *unique*. The template's *complexType* comprise of sequences of elements with attributes definitions and represent the whole relational schema (represented by relational directed graph) except the *PK*, *FK* set and μ attribute. The *PK*, *FK* set and μ attribute will be mapped to XML Schema's constraints components. The algorithm is described as follows:

(1) Map each table node in the relational directed graph to a *complexType* entity and let the *complexType* name'value be the value of the table node.

(2) For each table node, map its sub-node(s), or column node(s), to one or more <element> tags that describe the sub-elements of the complex element that is signed by the table name.

(3) For each column node, map its sub-nodes, τ and λ respectively to the *type* and *use* attributes of each element that the column node has been mapped to. So, after this step the whole relational schema except the *PK*, *FK* set and μ attribute will be mapped to several *complexType* entities.

(4) In this step we translate each primary key node, foreign key node and μ in the relational directed graph into XML Schema's *key*, *keyref* and *unique* constraints respectively. The only difference between key and unique constraints is that the unique constraints can accept the empty values but the key constraints cannot. If a column *c* is the only component of a primary key $pk=\{\{c\} \xrightarrow{pkey} t\} \in PK$ it will also has the unique attribute, under the circumstances we will ignore the unique attribute and only map that key node to XML Schema's *key* constraint.

We refer reader to our previous work [4] for further discussion about this subject where necessary.

After the Static Transformation, there will result in two layer data, one is the bottom relational data, and the other is the virtual XML view over the bottom data. Other systems can directly import or query these views to use data.

3.2. Dynamic query tier

In real applications, users need to query the virtual XML dynamically and acquire resultant information in XML format. But all data is stored in bottom database, so the users' query on virtual XML view should be transformed into the SQL query on the bottom database, and the query resultant tuple, which should be tagged into XML format as they are returned. The whole procedure is shown in figure 4, where label 1 means the Static Transformation between XML and RDB introduced in 3.1, label 2 means get query on XML view from user interface, label 3 means the transform from XQuery into SQL, label 4 denotes SQL execution on RDB, label 5 means outputting the result tuple and label 6 means adding tags in the executed result and return it back to the user interface in XML format.

Exiting XQuery-to-SQL methods all depend on the XML-To-RDB mapping algorithms. But in the real

applications, an independent method is needed because people can't ensure that the two translating steps come from the same industry projects thus conflicts can't be avoided. Aiming at this problem, we propose an XQuery-to-SQL translating method that is independent on any mapping algorithm from XML to RDB. As shown in figure 5, the method founds a four-tier structure to realize XQuery-to-SQL. The first tier gets input XQuery from user interface and equivalently simplifies it to one of forms that is suit for translation to SQL. The second tier defines a virtual general relational schema and translates simplified XQuery to SQL on it. The third tier translates SQL on virtual general relational schema to SQL on real storage relational schema by constructing a view of it. The fourth tier optimize the created SQL by reduce joins and group the result tuple. Our method has several advantages. First, it separates XQuery-to-SQL method from XML-to-RDB algorithm. Second, it improves the XQuery-to-SQL algorithm by optimizing.

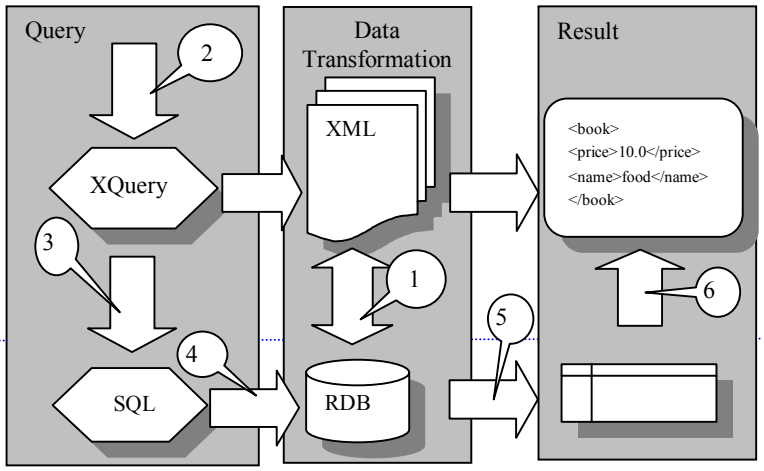

Fig 4: The dynamic query tier.

Shown in figure 5, to realize an XQuery-to-SQL method separated from XML-to-RDB we found a virtual generic relational schema, which can represent any XML document. Firstly, we get XQuery from user interface and simplify it to one form that is suit to translate to SQL. In succession, we translate XQuery on the XML documents to SQL on the virtual generic relational schema; at same time we create the tagging template. And then we translate SQL on the virtual generic relational schema to SQL on the real storage relational schema by defining a view on it. This view is a bridge between the two schemas. At last some optimization work is done to improve the method. The optimization SQL is executed on the real storage relational schema and return tuple. We use the created tagging temple tag the tuple and return the result in XML format.

From the viewpoint of relational theory, any XML document contain entities such as document, Element, Attribute, URI, Namespace, Comment, Qname, their Value, Child and we give another entities TransClosure. So the generic, virtual, relational schema that we use is shown in figure 5.This schema is constructed as a fully normalized relational version of the hierarchical structure of an XML document; note the use of foreign keys (shown in bold) to represent the filiations of different entities within a document.

Using the virtual generic relational schema has several advantages. First it separates the logical and physical views of the data, therefore it frees us from having to provide a separate XQuery-to-SQL translation algorithm per XML-to-SQL mapping. Second, by being so close to the structure of a document, it also offers good support for query translation. Third, examining the translation process on such a simple and generic structure, we can assess which XML query language constructs are feasible within a RDBMS independently of the mapping schema, which are not.

The Key Module in the Architecture:

1) XML simplify module: This module gets XQuery from user interface including programs or men and applies simplify rules to bring the user XQuery to one of the forms that we are able to translate to SQL.

2) Translator from XQuery to SQL: This module incrementally applies the XQuery to SQL translating rules given in the full paper. The rules translate XQuery constructs to SQL over a virtual relational schema.

3) SQL rewriting to real storage schema: in this module, using definitions of data as views over the virtual schema, the rewriter outputs a query that can be executed by the relational engine. We show in

section 7 what modifications we suggest for simple relational rewriting algorithms to deal with the kinds of queries that we produce.

4) Optimize SQL: in this module, we will give some optimization rule to optimize the SQL by reducing joins and grouping the result.

5) Tagger process module: this module is in charge of the construction of new elements that constitute the output of the query. To describe the structure of the result, a tagging template is computed during the translator to the tagger

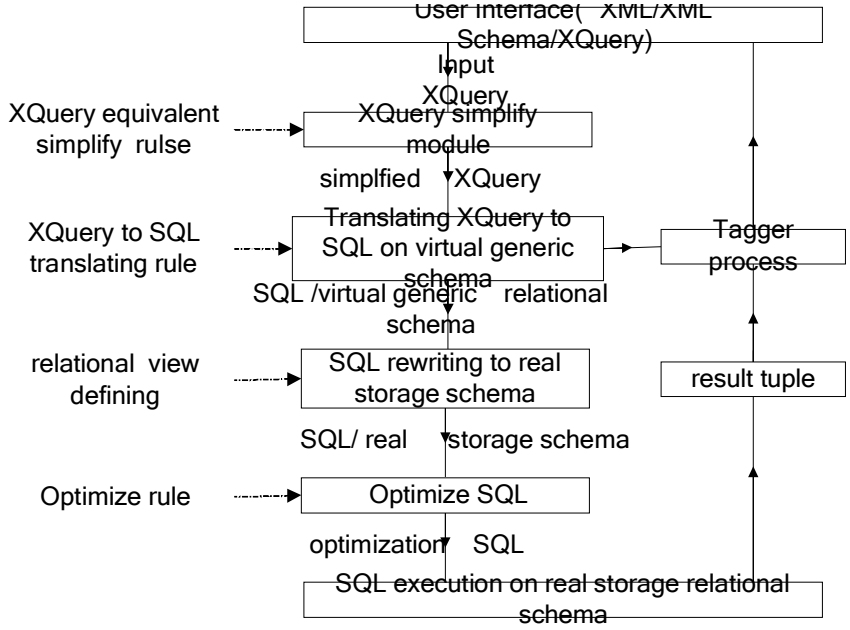

Fig 5: Four-tier architecture for translating XQuery to SQL.

We do not attempt to take further discussion about this subject in this paper, but we refer reader to our previous work [7] for any detail.

3.3. Synchronous updating tier

When a user application need to update data, the only updating on virtual XML files isn't enough, we must ensure the synchronous updating on bottom RDB with that on the virtual XML files. Therefore, the XML updating command should be mapped to RDB updating one and the two executions should be strictly synchronous. First of all, we parse and conclude the XML updating commands, and transform them into SQL with validity verified, then XML structure is updated as SQL implemented in RDB.

We propose a synchronous updating method between XML View and RDB. Our main contribute is as follows. Firstly, for there is no standard XML updating language such as XQuery for XML query, we give some standard update operations on XML view and translate these update operations on XML view to SQL on Bottom RDB. Then, we realize the Synchronous between updating XML view and updating Bottom RDB. We have implemented this update approach in our previous work [8]. For further discussion about this subject, we refer reader to [8].

4. The XMLRDB-BRIDGE

To realize the above bi-directional integration between XML and RDB, we design the middleware XMLRDB-Bridge locating between XML files and RDB, which manipulates the bottom database through JDBC on one side and offers or manipulates data in XML format for the user applications on the other side. Corresponding to the three tires of the integration technology, XMLRDB-Bridge constitutes of three functions: a) transforming data between RDB and XML, b) dynamically converting users' XQuery query on XML to the SQL query on RDB and return the result in XML format, c) synchronously updating RDB with the updating of XML to ensure the consistency between XML and RDB. Of all the three functions, a) is their basis and the transformation medium. XMLRDB-Bridge gets the relational schema through the reverse design on the RDB, and then transformation function is implemented between this relational schema and XML-Schema. The whole structure of the middleware is shown in figure 6.

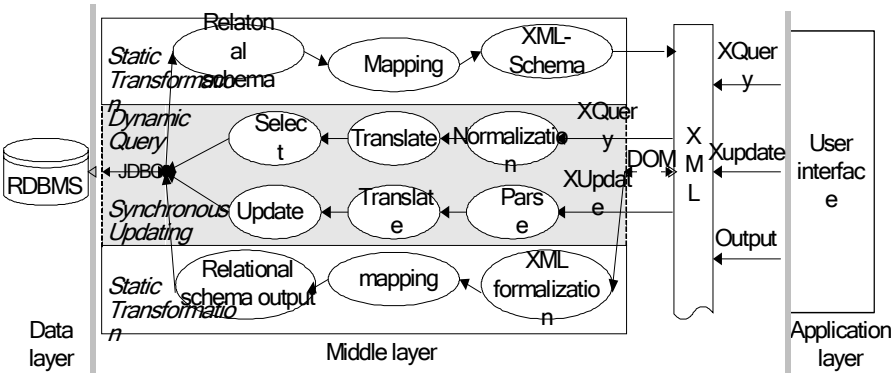

Fig 6: The architecture of the middleware between XML and RDB.

In technology aspects of realization, XMLRDB-Bridge takes the technique of JDBC (Java Database Connect) as the database interface, takes J2EE as the platform. Besides, to ensure the synchronous updating of the database, the middleware assigns a special thread to visit database and a full time listener to monitor the changes of the XML file. Once the listener finds the XML file is updated by the user applications, the thread is started to manipulate the database.

5. Conclusion

The application based on bi-directional integration between XML and RDB for provision of effective semi-structure management and usage is promising. However it needs more industrial case studies to be carried out in order to evaluate finally the utility. Many relevant aspects of a complete semi-structure data management environment are not addressed in this paper so far. We will show more details in our further work.

6. References

[1] R, R. Bordawekar, C. A. Lang. Analytical Processing of XML Documents: Opportunities and Challenges. *SIGMOD Record*, 2005, **34**(2).

[2] K. Michel. XML, RDF, and Relatives. *IEEE Intelligent Systems*, 2004, **16**(2): 26-28.

[3] H. W. Sun, S. S. Zhang, J. T. Zhou, et al. Constraints-preserving Mapping Algorithm from XML-Schema to Relational Schema. *Lecture Notes in Computer Science*, 2002, **2480**: 193-207.

[4] J. T. Zhou, S. S. Zhang, et al. An XML-Based Schema Translation Method for Relational Data Sharing and Exchanging, *Proceeding of CSCWD*, Xia Men, 2004.

[5] P. V. Biron, A. Malhotra. XML Schema Part 2: Datatypes, http://www.w3.org/TR/xmlschema-2, 2001

[6] D. W. Lee, M. Mani, et al. Nesting-based Relational-to-XML Schema Translation. *Int'l Workshop on the Web and Databases (WebDB)*, Santa Barbara, 2001.

[7] H. W. Sun, S. S. Zhang, J. T. Zhou, et al. XQuery-to-SQL Translating Algorithm with Little Dependence on Schema Mapping Between XML and RDB. *Proceeding of CSCWD*, Xia Men, 2004.

[8] H. W. Sun, S. S. Zhang, J. T. Zhou, J. Wang, H. Zhao. Synchronized Update in Data Integration Between XML and RDB Source. *Journal of Northwestern Polytechnical University*, 2004, 22(3): 333-337.