VXMLR: A Visual XML-Relational Database System

Aoying Zhou *Hongjun Lu** *Shihui Zheng*
Yuqi Liang *Long Zhang* *Wenyun Ji* *Zengping Tian*

Fudan University, Shanghai, China, ayzhou@fudan.edu.cn
* Hong Kong University of Science & Technology, Hong Kong, China, luhj@cs.ust.hk

Abstract

We demonstrate a visual based XML-Relational database system where XML data is managed by commercial RDBMS. A query interface enables users to form path expression based queries against stored data visually. Statistics about data and a special path directory are used to rewrite path expression based queries into efficient SQL statements involving less number of joins.

1. Introduction

Anticipating that a large number of XML data will be available, various approaches to storing and querying XML data have been proposed. There are basically three alternatives: storing XML data in semi-structured repository, in object-oriented databases, and in relational systems. This demonstration presents VXMLR, a visual based XML document management system built on top of a commercial relational database management system.

Figure 1 depicts the architecture of VXMLR. An input XML document is first parsed into a DOM (Document Object Model) tree. At the same time, the DTD (Data Type Definitions) for the document is extracted. The document tree is then mapped into relational tables and stored in the database. Both the DTD structure and mapping information are maintained in a DTD Directory, that is used in query rewriting and result construction. To access XML data stored in the database, VXMLR supports a visual querying interface. Through the interface, DTD structures of stored XML documents are displayed, and users can form queries by clicking the relevant data elements and entering conditions. Such queries are first expressed as path expression queries that are then transformed into SQL statements ready to submit to the underlying relational DBMS. To generate efficient SQL statements from path expression queries, VXMLR maintains some statistics of data and a path directory, which are used in the query rewriting process to reduce the number of SQL statements and simplify join conditions. The returned query results are constructed and expressed using XSL before being delivered to the user through the querying interface.

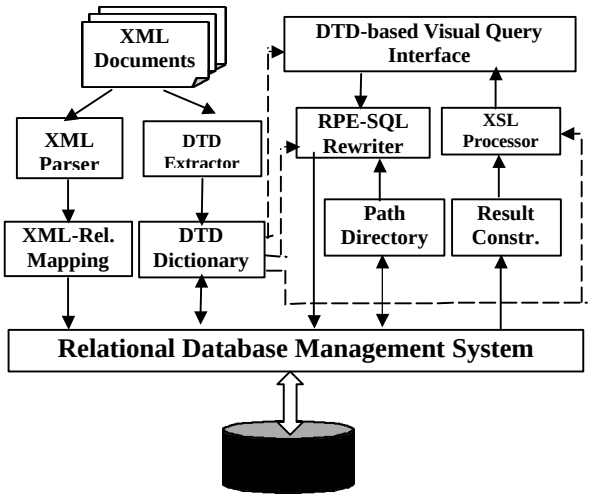


Figure 1: The architecture of VXMLR

2. Querying XML data in relational systems

An XML document usually has its structure defined by its DTD (Data Type Definition). Such structure is more complex than relational tables. When an XML document is stored in relational systems, it is flattened into relational tables. Furthermore, SQL, the main query language supported by relational systems, has to be used to query the stored data. Obviously, impedance mismatch problem occurs in such a system. Therefore, we concentrate ourselves on efficiently querying XML data stored in relational systems when developing VXMLR. In particular, two main issues are addressed: ease of forming queries and efficient retrieval of stored data.

Querying XML data is difficult since users need to know the structure of the data. When XML data is stored

Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001

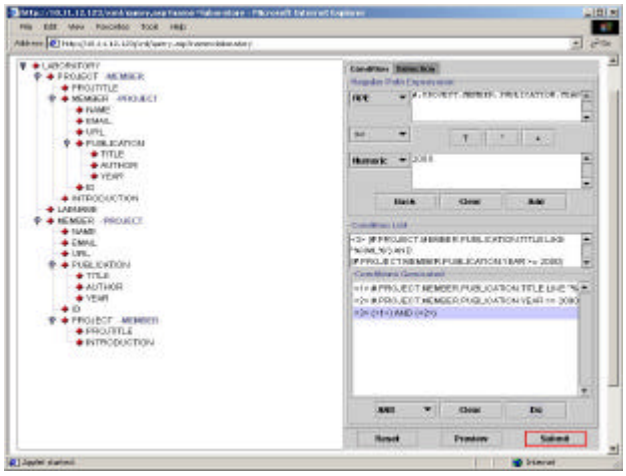
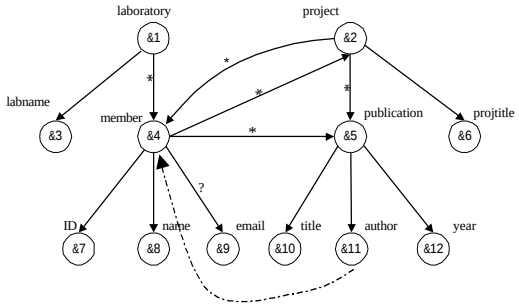


Figure 2: The VXMLR query interface



Relational Schema:

laboratory(ID, labname , PARENTID);
project(ID, projname, PARENTID);
member(ID, name, email, PARENTID);
publication(ID, title, author, year, PARENTID);

Figure 3: A Sample DTD and its relational schema

in relational systems, this becomes more difficult as the original structure may not be well reflected by the relational schema. VXMLR implemented a visual query interface as Java Applet running by browsers at client sides. Figure 2 is the screen dump of the interface. It consists of two portions (windows). The left portion displays the DTD structure of the XML document to be queried. The right portion is for user to specify the target attributes to be retrieved and the conditions to be satisfied by the retrieved data items. Users can simply click the data elements in the DTD portion to select them and enter the conditions at the right hand side windows. After a user completes the specifications of the target list and conditions, a query in the form of path expressions is generated.

A path expression has the form

$r = (r)^* \mid (r)^+ \mid (r)? \mid r_1.r_2 \mid r_1|r_2 \mid \# \mid name$.

where *, +. ? mean 0 or more, 1 or more, and 0 or 1 occurrences, respectively. Concatenation $r_1.r_2$ is used to form a path from r_1 to r_2. Alternation "|" stands for disjunction. The # sign denotes arbitrary occurrences of any regular expressions. We distinguish two types of path expressions: *simple path expression* (SPE) and *regular path expression* (RPE). SPE are path expressions that consist of only element or attribute names. For example, with the data described in Figure 3, a SPE query that retrieves the name of all members who have publications,

```
select member.publication.author.name
```

can be rewritten into the following single SQL query in a rather straightforward way:

```
select m2.name
from   member m1, publication, member m2
where  publication.perantid = m1.ID
and    publication.author = m2.ID
```

Note that the number of joins in the result SQL query is equal to the number of intermediate nodes on the path. To minimize the number of join conditions, VXMLR maintains a path directory, which is similar but more than join indexes that materializes the paths from the root to elements existing in the data. Therefore, instead of join conditions, path directory entries can be used in the SQL statement, which reduces the number of join conditions, hence improves the performance, dramatically.

Regular path expression queries (RPE) that contain "#" and "*" need to be expanded to SPE queries first, then translated into SQL statements. For example, query

```
select project.#.publication
```

selects all of the publications reachable from the project node via zero or more edges. With the information in DTD, the # in the query is expanded and the query becomes

```
select project.member.(project.member)*.
                                publication
|project.(member.project)*.publication
```

When expanding the * operator, the number of result SPEs depends on the number of occurrences of paths involved in the operator. Assuming that *project.memer* occurs on each path at most once in the data, the above RPE can be expanded to :

```
select project.member.publication
union
select project.publication
union
select project.member.project.publication
```

To facilitate this expanding process to only produce necessary SQL statements, i.e., statements that will return some results from the current data, VXMLR maintains statistics on the cycles in the data graph and developed algorithms to effectively use such statistics.

3. Conclusions

The system was implemented on top of Microsoft SQL Server. The server side program is implemented in C++. The client side program, i.e., the visual query interface, which runs on a browser, is written in Java. Experimental study indicted that the system is user friendly as well as performs well with various types of data.