XML Data Mediator
Integrated solution for XML Roundtrip from XML to Relational

Nianjun Zhou
IBM
150 Kettletown Road
Southbury, CT 06488
+1-203-486-7435
jzhou@us.ibm.com

George Mihaila
IBM
T.J. Watson Research Center, PO 218
Yorktown Heights, NY 10598
+1-914-784-7803
mihaila@us.ibm.com

Dikran Meliksetian
IBM
150 Kettletown Road
Southbury, CT 06488
+1-203-486-5590
meliksd1@us.ibm.com

ABSTRACT
This paper presents a system for efficient data transformations between XML and relational databases, called XML Data Mediator (XDM). XDM enables the transformation by externalizing the specification of the mapping in a script and using an efficient run-time engine that automates the conversion task. The runtime engine is independent from the mapping script. A parser converts a mapping script into an internal conversion object. For the mapping from relational to XML, we use a tagging tree as a conversion object inside the runtime engine, and use an SQL outer-join scheme to combine multiple SQL queries in order to reduce the number of backend relational database accesses. For the mapping from XML to relational, the conversion object is a shredding tree, and we use an innovative algorithm to process the XML as a stream in order to achieve linear complexity with respect to the size of the XML document.

Categories and Subject Descriptors
E.2 [**Data Storage Representation**]

General Terms
Algorithms, Performance, Design

Keywords
XML, RDBMS, relational database, shredding, XML, XSL

1. INTRODUCTION
The XML Data Mediator (XDM) is a tool for bi-directional data conversion between XML and structured data formats of relational databases or LDAP. We abstract the concepts of relational database and LDAP into a concept called data storage. Mapping is defined at the data storage and XML schema (or structure) level. XDM supports the transformation between XML to multiple data storages (including databases and LDAP repositories).

XDM externalizes the specification of the mapping, and it replaces the programming effort by the simpler effort of writing a script that describes the relationships between the XML constructs and the corresponding relational constructs. XDM can be used as a stand-alone utility, or it can be integrated as a library in other applications, such as middleware.

The XDM runtime engine is designed to accept an input script as a template for performing transformations. The runtime engine parses the script only once and creates efficient run-time objects that are cached and reused as needed. Based on the script and the direction of the transformation (from data storage to XML or from XML to data storage), XDM creates the appropriate tree structures for performing the transformations. The tree structures define both the data storage procedures to be executed as well as the XML structure information, which is either an XML hierarchy to be created upon retrieval or to be traversed for storage.

Included in the current implementation are two types of scripts: Document Type Definition with Source Annotation (DTDSA) [1] and XML Relational Transformation (XRT). The DTDSA script is a DTD marked with mappings between elements and rows/columns in the database. The same DTDSA script can be used for both directions of the transformation. The XRT script is a script loosely based on XSL which uses an SQL query-based approach to obtain the elements. Since a given script is likely to be used for multiple transformations within the lifetime of an application, the run-time engine was designed with a number of caching mechanisms. The run-time engine parses the script only once and creates efficient run-time objects that are cached and reused as needed.

Currently, XDM is implemented in Java, and we are using JDBC to access relational databases and JNDI to access LDAP repositories. Therefore, the implementation is vendor-neutral, and can support all the relational databases and LDAP repositories.

2. FROM RELATIONAL TO XML
Tagging trees are generated by a script parser and used to facilitate transforming data from relational to XML. Tagging trees represent both the XML hierarchical structure information as well as the query information to retrieve the data from database and to place in the hierarchical structure. A runtime environment then processes the tagging tree by a depth first traversal. The runtime environment is able to be configured to output a hierarchical XML document, or pipelined to control, for example, SAX processing.

There are multiple node types has been defined for a tagging tree to catch the features of the transformation from relational data to

Copyright is held by the author/owner(s).
WWW 2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

XML. Based on the behavior at runtime, the nodes can be categorized as non-walk-able node and walk-able node. The non-walk-able nodes include attribute data nodes, text data nodes and binding nodes. A binding node is a special type of node, which specifies the relationship between a data node and an execution node. Walk-able nodes include element nodes and execution nodes. Element nodes contain an element name and have as children one data node, zero or more attribute nodes and zero or more element nodes. An execution node contains a parameterized SQL command. The output of the execution of the contained parameterized SQL command is a dataset.

The runtime execution itself has two steps: 1) Information from the database is retrieved using the SQL queries in the execution nodes, thereby creating an array of intermediate data sets; and 2) The data from the intermediate data sets is transformed into the required final output XML format using format specified by the tagging tree.
One of the factors affecting the performance of the operations is the number of backend database accesses. The problem becomes worse if the backend database resides on a remote machine. In order to reduce the number of database accesses, we developed a scheme to combine multiple queries together using the SQL outer-union construct. Our outer-union operation in query combination differs from the conventional SQL union operation in some respects. The conventional SQL union operation is only defined for relations having the same number and types of columns. Each query optimization combines a set of queries where each query corresponds to an execution node in the query tree. The first execution node that contains queries to be merged into an outer-union query will be replaced with an outer-union execution node, which contains the generated outer union query, and with an associated virtual execution node. Virtual execution nodes replace the other execution nodes on the selected path of nodes specifying the merged queries. The output columns of each virtual node are connected to the corresponding columns of the output of the outer-union execution node.

3. FROM XML TO RELATIONAL

In the opposite shredding or storage process, XDM creates a shredding tree, which defines the appropriate XPath expressions used to select the proper elements from the XML input, and the data storage procedures, in order to place them in the defined data storage locations. The data storage operation could be an SQL update statement (INSERT, UPDATE, or DELETE) or complex data manipulation.

A shredding tree has two types of nodes: cursor nodes and data nodes. All of the nodes of the shredding tree are labeled with hierarchical locators. A hierarchical locator is a path expression obtained by concatenating the labels of the group nodes from the root to a node in a schema tree. Each shredding tree has a local lookup table, which maps a node locator to its corresponding shredding node. A node locator can either correspond to a cursor node or data node, but not both.

At run time, we maintain multiple virtual tables in memory to hold the data from XML document. The runtime engine will

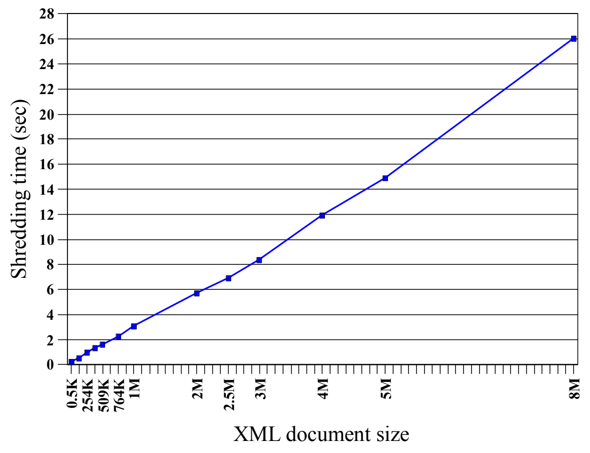

Figure 1. Shredding Time

maintain certain counters for each virtual table, recording the number of columns for which values are still expected. As soon as all the counters become zero, all the columns of the current record of virtual record are filled in and ready for shredding. This shredding method is a linear-time algorithm for shredding of XML into relations. The main advantage of our algorithm is the fact it processes the input XML document in a streaming fashion (by using a SAX parser) and therefore does not need to materialize the entire XML tree in main memory. As a direct implication, it scales nicely to arbitrarily large document as shown in Figure 1.

4. CONCLUSION AND FUTURE WORK

We have presented an integrated, vendor-independent solution of mapping between XML and relational databases and LDAP. The XDM system can be used in a variety of applications, including Web Content Management systems, Knowledge Management systems, and Business-to-Business transactions. Wherever there is the need to extract selected XML elements and to store them in a data sink, or to compose an XML object from various data sources, XDM can improve the efficiency and the scalability of any application.

Some open issues involve how to shred recursive XML documents into relational database. These are practical problems encountered in catalog business and life science areas. Besides, how to use the statistical data usage of a database to provide optimization guideline for query combination is still an open question.

5. ACKNOWLEDGMENTS

Many thanks to Sriram Padmanabhan, Colleen Viehrig, Arun Pradeep, Jessica Wu Ramirez for their contributions to XDM development.

6. REFERENCES

[1] Ming-Ling Lo, Shyh-Kwei Chen, Sriram Padmanabhan, Jen-Yao Chung: XAS: A System for Accessing Componentized, Virtual XML Documents. ICSE 2001: 493-502