

Querying XML documents

Taking the hassle out of retrieving web-based information

John A. Miller and
Sonali Sheth

Getting answers to those
burning questions

©PhotoDisc/Composite: D. Cantillo

Web-based environments will typically need to maintain many documents (e.g. simulation model information and data results from multiple sites). Ideally, this information would be held in a widely utilized and highly viewable format. The eXtensible Markup Language (XML) is a new standard that meets this criteria for world-wide web based applications. It is sophisticated enough so that complex real-world structures and relationships may be captured. Large collections of XML documents can be stored for efficient retrieval using Database Management Systems (DBMSs).

Current database management is represented by the Object-Oriented (OO) and Object-Relational (OR) database systems. Each type has their own query languages, Object Query Language (OQL, v.2) and Structured Query Language (SQL), respectively.

These database query languages provide a high-level way to ask for information from a database. For example, a concise query in SQL may be equivalent to hundreds of lines of code in a programming language with a low-level (e.g., find/get record style) Appli-

cation Programming Interface (API). However, while the original SQL was quite easy to learn, the same cannot be said for SQL3.

Also, even though both database systems can store XML documents, these documents do not conform to either database model. Rather, XML documents conform to a more flexible semi-structured data model. Hence, a few problems exist: 1) the OQL and SQL3 query languages are not ideal for accessing semi-structured documents, 2) they lack sophisticated information retrieval capabilities and 3) they require substantial time to learn.

This article addresses storing and retrieving XML documents. It surveys work being done on query languages and tools for XML. It then discusses a simple graphical XML query tool for front end use in the Java Simulation (JSIM) web-based simulation environment.

XML as the universal form

Information systems have many applications and subsystems that must work together to support the information needs of organizations. In the past, it has been very difficult to get various,

independently developed software to interoperate. Fortunately, recently developed applications and subsystems have one thing in common, they all utilize objects (e.g., OO Programming Languages such as Java and C++, OO and OR Databases, and middleware such as CORBA and DCOM). Unfortunately, the form of object can vary so an object has to be mapped to all possible forms. It would be better to have one form as the *standard for data interchange*. Several companies seem to believe that XML will be that standard.

In general, XML would serve as an interface layer or wrapper for data being passed between data sources. This would make it possible for a wide variety of applications, legacy systems and databases to exchange information. Thus, XML would become the web-style Electronic Data Interchange (EDI) standard. Let us look at some of the possibilities for data exchange with XML as the common format.

Java. Java and XML are very complementary. Java can dynamically produce XML documents. XML forms may send data to Java Servlets. To facilitate the storage of Java objects in a form other tools and systems can understand, Java objects in main memory can be made persistent (stored in stable storage, e.g. on disks) as XML documents (or parts of an XML document). This would replace the current conversion to binary files as is currently done when Java objects are serialized.

Web browsers. An XML document should be thought of as data first and as a displayable document like Higher Text Markup Language (HTML) page second. The display rendering information is given separately as an eXtensible Style sheet Language (XSL) specification. Currently, few browsers display XML documents, but many more are expected to become available in the near future. Available browsers include Jumbo, Fujitso HyBrick, Netscape Communicator 5 source code release and Internet Explorer 5.

Object-Relational Database Management Systems (ORDBMSs). XML documents may exhibit complex structures including deep nesting. Thus, traditional relational databases are not ideally suited for storing these documents. Object-relational databases augment relational databases with object-oriented capabilities making them more suitable. But as mentioned earlier, most object-relational database

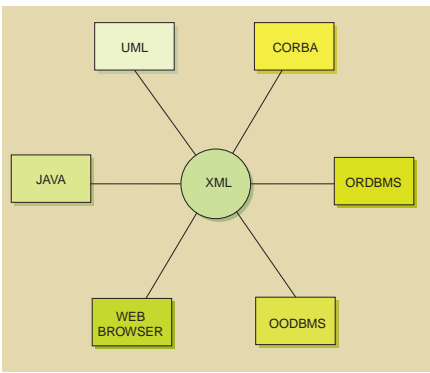


Fig. 1 XML as a universal data object

systems currently implement a subset of the SQL3 standard.

One possibility is Oracle 8i which is Oracle's object-relational database for Internet computing. It supports information definition and access through XML. Oracle sees XML as defining all resources in one place, regardless of type and location of the information. Oracle 8i uses structures in XML to build database structures and store data or links to data (through locators). This is done according to the rules and constraints defined in XML. Individual XML instance objects are decomposed and stored according to the definitional structure. Upon access, the data is pulled and the XML object reconstructed and presented through the browser.

Object-Oriented Database Management Systems (OODBMSs). Object-oriented databases are designed to handle objects in their native forms. The objects then maintain their own data, methods, relationships and semantics of the whole model. This is ideal for the creation and management of hierarchical XML trees. Also, data from such a database does not need to undergo transformation to be used at the client side; XML handles all of that. A couple of object-oriented databases that provide early support for XML are Object Store and Poet.

Object Store, an object-oriented database, can store tree-like structures as is. This eliminates the mapping code often required to convert the hierarchically structured XML object into rows and columns. With Object Store, navigating an XML tree is as simple as traversing a pointer in local memory. Object Store also has index and query capabilities that facilitate high-speed queries. Object Store also boosts performance by supporting use of dynamic data. It can support a data abstraction model for XML. This allows developers to add a new field or attribute to a data structure without having to redefine the object model

or evolve the schema.

When used as a cache for application servers, the transactional and data consistency is managed by the Cache-Forward architecture. Object Store with its ability to manage, serve, index and query native XML data, thus provides a good foundation for many data integration solutions.

Poet Object Server addresses the needs of XML as well as the needs of associated applications. It stores the XML elements intact, along with the structural semantics, thus maintaining the integrity of the XML model. Poet provides an object-oriented database that can be scaled up as well as down while leveraging the same APIs. This simplifies application development and provides a language-independent object model. Thus, developers can mix and match C++ and Java even in a single application. Poet offers simple standards-based APIs and embeddability. It also provides for a higher granularity locking, down to element level. This is critical for user scalability.

Common Object Request Broker Architecture (CORBA). CORBA is a major standard for distributed object computing. Most CORBA products provide a mechanism for making CORBA objects persistent. For maximum interoperability, the states of CORBA objects may be saved as XML objects in documents. Also different data types can be mapped to data types in XML with some modifications.

Unified Modeling Language (UML). UML is positioned as a general-purpose, unified, graphical modeling language for primarily object analysis and design. Its main components are a meta model and a notation for graphically representing the concepts from the meta model. It provides one language to the software and system development community.

With XML gaining momentum throughout the industry, efforts have begun to integrate existing meta models and meta modeling with XML. Design tools based on UML can generate code or specifications in several languages and formats.

Some work is being carried out by the CASE Data Interchange Format (CDIF) Technical Committee. CDIF is a Family of Standards that lays out a single architecture for exchanging information between modeling tools, and between repositories. CDIF also defines the interfaces of the components to implement this architecture.

The XML-based CDIF syntax allows the exchange of meta models and models using the emerging XML standard.

Converting XML to database schema

If XML documents are required to have a Document Type Definition (DTD), they are called valid XML documents. (When no DTD is given, the document just said to be well formed.) It is straightforward to convert a DTD into an object-oriented schema specification. For example, the following DTD can be readily converted to Object Definition Language.

Consider the Fig. 2 and Fig. 3 XML document type definitions for office MEMO and MEETING document types, adapted from an XML tutorial. Each MEMO and MEETING definition contains an ID type attribute to uniquely identify each memo/meeting. The attribute MEMO-REFS in MEETING is the list of memos discussed in that meeting. The attribute MEETING-REFS in MEMO is the list of meetings in which the memo was discussed. The document type definition for MEMO is shown in Fig. 4.

Forming queries from DTDs

Patterns may be substituted into a DTD to form the basis for a query. Such a query can be run against one or more databases. All matching XML documents will be retrieved for which the conditions in the query are true. This form of querying a database is analogous to the approach used in Query-By-Example (QBE). With a graphical tool supporting

```

<!DOCTYPE MEMO [
  <!ELEMENT MEMO (TO+, FROM, SUBJECT, BODY) >
  <!ATTLIST MEMO MEMO_NUM ID #REQUIRED >
  <!ATTLIST MEMO MEETING_REFS IDREFS #IMPLIED >
  <!ELEMENT TO (#PCDATA) >
  <!ELEMENT FROM (#PCDATA) >
  <!ELEMENT SUBJECT (#PCDATA) >
  <!ELEMENT BODY (P+) >
  <!ELEMENT P (#PCDATA) >
] >
  
```

Fig. 2 Memo document type

```

<!DOCTYPE MEETING [
  <!ELEMENT MEETING (ATTENDEE, PLACE, CALLED BY) >
  <!ATTLIST MEETING MEETING_NUM ID #REQUIRED >
  <!ATTLIST MEETING MEMO_REFS IDREFS #IMPLIED >
  <!ELEMENT ATTENDEE (#PCDATA) >
  <!ELEMENT PLACE (#PCDATA) >
  <!ELEMENT CALLED BY (#PCDATA) >
] >
  
```

Fig. 3 Meeting document type

```
interface MEMO
(extent MEMOS
key (FROM, SUBJECT))
{
attribute set<string> TO;
attribute set FROM;
attribute set SUBJECT;
attribute set<string> BODY;
}
```

Fig. 4 Document type definition of Fig. 2 as expressed in Object Definition Language

specification to

```
<!!ELEMENT FROM ( " J o h n
Miller" | "John A. Miller")>
If one is interested in memos from
any Miller, the FROM of Fig. 5 is
changed to
<!!ELEMENT FROM ("_*Miller")>
```

XML query tool

Consider expressing the following complex query using the graphical XML query tool shown in Fig. 6.

Query: Find the memos where the person who received the memo is the person who called the meeting for discussing the memo and whose name ends with "Miller."

This query involves selecting and placing query boxes on a canvas and then linking them together. Clicking on a query box brings up a dialog box in which the name of a DTD is given. Pressing the get button will place the DTD in the dialog box.

Text boxes may be modified to constrain the elements. Check boxes are provided for each element. Checking an element selects its value to be displayed in the output of the query result. Finally, conditions that apply to two different DTDs are specified in a join box associated with the link between the two query boxes.

Summary

XML is likely to play a central role in future information systems. It will be used for data interchange and XML documents will need to be stored in databases.

With knowledge of XML, complex queries can be formulated without resorting to using traditional queries. Such a simple form of querying will be very useful to those not inclined to learn database query languages.

Read more about it

- Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. <<http://www-db.stanford.edu>>.
- Introduction to cdif <<http://www.eia.org/eig/cdif/intro.html>>.
- Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy and Dan Suciu. XMLQL: A Query Language for XML. 1998. <<http://www.w3.org/TR/1998/NOTE-xml-ql19980819>>.
- Metamodel information. <<http://www.metamodel.com>>.
- J.A. Miller, Y. Ge, and Z. Tao. *Component-Based Simulation Environ-*

ments: JSIM as a Case Study Using Java Beans. pages 786-793, December 1998.

- *Server-side xml: Taming the tower of Babel.* <<http://www.odi.com/ODILIVE/contents/happenings/tech-trends/whitepapers/ServerSideXML.pdf>>
- Internet database solutions with oracle 8i. <http://www.oracle.com/html/webdb.html>.
- *XML-the foundation for the future.* <<http://www.poet.com/PDF/xml.pdf>>.
- Jonathan Robie, Joe Lapp, and David Schach. *Xml query language (xql).*
- Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems.* Prentice Hall, Upper Saddle River, New Jersey.
- A tutorial in XML and XSL authoring. <http://pdbeam.uwaterloo.ca/rlan-der/XML_Tutorial/>.
- Query By Example. NCC, AFIPS, page 44,1975.

About the authors

John A. Miller is an Associate Professor and the Graduate Coordinator in the Department of Computer Science at the University of Georgia. Dr. Miller received the B.S. degree in Applied Mathematics from Northwestern University in 1980 and the M.S. and Ph.D. in Information and Computer Science from the Georgia Institute of Technology in 1982 and 1986, respectively. He is an Associate Editor for *ACM Transactions on Modeling and Computer Simulation* and *IEEE Transactions on Systems, Man and Cybernetics* as well as a Guest Editor for the *International Journal in Computer Simulation* and this Feb./March issue of *IEEE Potentials*.

Sonali Sheth is a member of the technical staff of NightFire Software, Inc., a telecommunications company in Berkeley, CA. She received her M.S. in Computer Science from the University of Georgia in 1999. Her areas of interest include telecommunications, database systems and Internet technologies such as Java and XML.

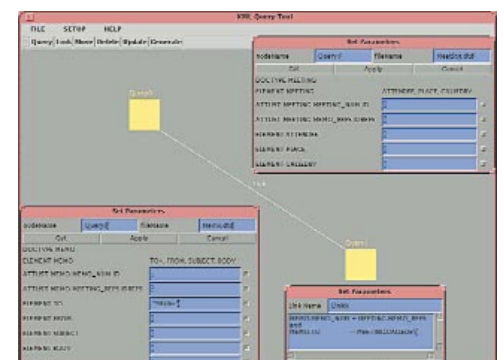


Fig. 6 XML query tool

Fig. 5 Query to find John Miller's memos

such an approach, finding information becomes a process of rapid query formulation. The search is then followed with browsing the query results and/or possibly reformulating the query. From a simulator's point of view, it is similar to surfing the Web. However, the query tool would provide much better precision than today's search engines.

Pattern matching

Pattern matching in XML documents is a straightforward process. Consider the memo document type shown in Fig. 2. Suppose one wishes to obtain all documents of type MEMO from John Miller but only see the subject (not the body) of the memos? Instead of explicitly writing a where clause with a condition, (select ... from ... where—complex queries may be hard to express in this form), the DTD itself can be used to specify the where condition (Fig. 5). Note that a "!!" symbol before any attribute or element denotes a attribute or element to be shown (as in a select clause).

Using regular expressions

Simply finding XML documents that match constant strings is too crude to serve as a query language. Thus, XML includes several regular expression operators that can be very useful in queries.

- | —alternation (match one of the cases)
- * —closure (repeat 0 or more times)
- + —closure (repeat 1 or more times)

The query in Fig. 5 may miss certain memos if the middle initial was sometimes used. To cover both cases, an alternation may be used. This is accomplished by changing the FROM element