QL'98

Query Languages Workshop Position Paper

Noah Mendelsohn

Lotus Development Corp.

November 10, 1998

Abstract

This paper discusses the use of XML in conjunction with a variety of repositories and database systems, and outlines some of the requirements for an effective query language. Many of the more general issues relating to query languages have been discussed in the references for the workshop, and are not repeated here. This paper therefore concentrates on one aspect of the query problem that seems to require particular attention.

Background

XML'ssimplicity, generality and broad applicability can lead to conflicting assumptions about it's likely use. When considering query languages, it is important to first agree on the nature of the information to be queried, the location and structure of the underlying data, and the needs of the applications and users that will "consume" the results.

The thesis of this paper is that XML query must support retrieval from all of the following information sources:

1. An XML repository or other XML-enabled database containing a direct representation of the XML information in its natural tree-structured form.
2. A non-XML repository, such as a relational database, which stores information in a manner optimized to its own needs. In these situations, XML will be presented as a "view" of the underlying data.
3. Applications which source XML completely dynamically on demand.
4. Distributed networks involving combinations of all of the three sources listed above.

Case #1 seems to be best understood by the XML community, as it corresponds to direct storage of XML documents.

Case #2 is at least as important, because non-XML stores offer unique power and flexibility in sourcing information to XML. As an example, consider a relational database containing two tables: table #1 has a row for each employee in an organization, while table #2 has a row for each office. One good use of these tables is to produce an "org chart", a hierarchy with the company president at the top, and employees arranged below according to the management structure of the company. Using joins, office and phone information can be included from table #2. Ideally, this information would be presented in an "ORG CHART ML" format, for use with XML tools. The power of a relational database allows *the same information* to produce a completely different "OFFICE SPACE ML", arranged hierarchically by building number, floor, office, etc., including information about the employee in each office. The relational database is used to do what it does well: flexible late binding of information to meet application needs. A static repository containing an ORG CHART ML document cannot easily produce the office space document. Relational databases are not the only example of a data store which is optimized to a particular set of applications. Our company, Lotus, offers another example: Domino and Notes are optimized toward collaborative manipulation of structured documents and document fragments. Our competitors offer similar systems, and the object database community provides yet a different model. An Alta Vista search would be yet another example of a query system based on text retrieval. The query architecture must effectively support XML retrieval from all of these sources.

To understand case #3, consider a query to retrieve crew scheduling information for an airline. It's quite possible that such information is assembled dynamically by an application program on demand, and returned as XML. Similarly, a vendor supplying quotes for parts might run an elaborate algorithm to determine prices based on demand, available inventory, etc., then return the resulting price list in XML. XML queries should be useable to request the appropriate reports and to select information from them.

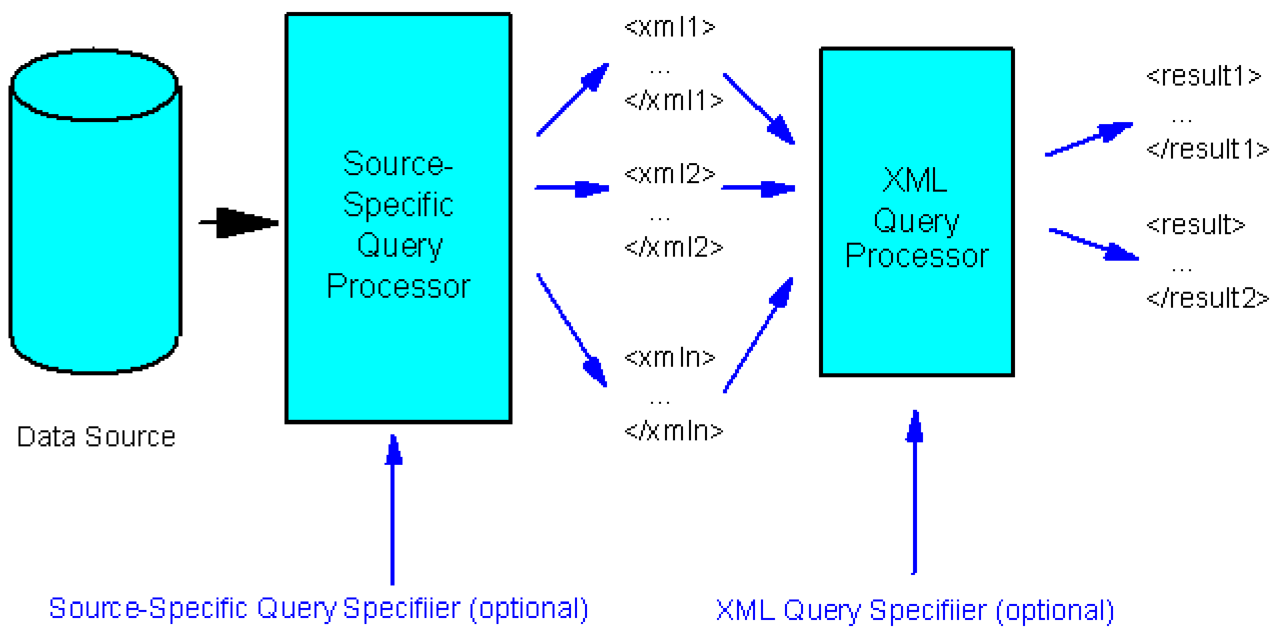
Case #4 involves searches across multiple sources. So, a query for prices on automobile parts might search several sites in parallel. One source might generate its quotes dynamically from an application, a second might retrieve information directly from a relational DB, while a third might have the price lists pre-stored in XML.

Position

A central position of this paper is that all the sources listed above must be supported in a natural manner by the query infrastructure, though not necessarily by a single closed query syntax. There is a natural tension between modeling all of these sources as if they were XML, and exploiting the unique capabilities of each.

It is difficult to imagine an XML-based hierarchical query that would describe the "joins" that produced both an organizational chart and a building map in the example above. Joins are best expressed in a relational query language such as SQL. For object databases, other languages such as OQL might be most appropriate. To access applications, an URL seems to be the most appropriate general abstraction.

This paper does not advocate any particular solution, but the following diagram seems to be a reasonable framework to consider:



The diagram illustrates the composition of a standard XML query with an optional source-specific query. In the examples above, the source-specific queries could be Alta Vista full-text searches, SQL or OQL queries, etc. The result of the source-specific processing is modeled as a forest of XML documents or fragments, against which an optional XML query can be executed.

Several refinements are possible depending on the circumstance. If the original data is already in XML, or sensibly modeled as XML, then the first stage query can be eliminated entirely (case #1); portability is maximized, at some cost in flexibility. Conversely, in the case where the result of the initial Alta Vista or SQL query (I.e. <xml1>, <xml2> ... <xmln>) is the desired result, then no XML query is needed. Finally, there will be situations in which the source specific query should be generated automatically from an XML query, or in which a source-specific optimizer would wish to combine the two for procesing.

In summary, an effective query architecture must embrace the native capabilities of a broad range of data sources, while leveraging the power of unified delivery and query refinement based on XML. One possible approach, using the web as an example, would allow for source-specific query specifiers to be prepended to the URL specifying an XML query. Either part would be optional, but in all cases, the result would be a forest of XML documents or fragments, typically offered through a standard API such as DOM.

Conclusion

This paper has explored one aspect of the XML query problem, the tradeoff between exploiting diverse data sources vs. unifying the approach to XML query. The tentative conclusion is that no single mechanism can exploit the power of full text, relational, application, and other data sources from which XML will be generated, but that an extensible query architecture, combining source-specific with XML-oriented mechanisms, might meet the needs of a broad range of applications.

Author Information

Noah Mendelsohn
Lotus Development Corp.
One Rogers Street
Cambridge, MA 02142 USA
Voice phone: 1-617-693-4036
Email: Noah_Mendelsohn@lotus.com