

# DATABASE

## TRENDS AND APPLICATIONS

Solutions for the Information Project Team • [www.dbta.com](http://www.dbta.com)

Volume 21, Number 1 • January 2007

# XQuery is Key to SOA Implementation

*Every service-oriented architecture should include a data architecture with XQuery and data services.*

**By Jonathan Robie and  
Brian Anderson**

Most articles on service-oriented architecture (SOA) give surprisingly little attention to data. In enterprise architectures, many SOA projects have been designed with great attention to how Web services are published, located, composed, and orchestrated, but with little attention to the infrastructure needed to efficiently provide and process the data these services depend on. Every service-oriented architecture should include a data architecture in which XQuery and data services play a key role.

In service-oriented architectures, services must efficiently process XML from requests, using a variety of data sources to create a response. For instance, a service might use data from the request itself, data stored in relational databases, XML files, and Web service calls, combining and restructuring this data to create the response. The service may need to manage XML input or output that is extremely large or complex. Web services depend heavily on XML processing and data integration, the very tasks for which XQuery was designed. Using XQuery simplifies these tasks dramatically for the programmer, and delivers significant performance improvements.

Many enterprises have not only multiple data sources, but multiple data consumers, such as dynamic Web sites, servlets, AJAX applications, SOA, and data publishing applications. Frequently, several applications require access to the same data, represented as

XML. Often, the code is recreated for each application, wasting programming resources and increasing the difficulty of consistent change control across applications. Applications can be simplified by creating data services, implemented in XQuery, that any application can invoke to obtain the XML data it needs.

### **XQuery Simplifies XML Processing**

SOA is based on XML, so using a query language based on XML makes things easier. XQuery was developed by the W3C, the same standards organization that developed XML, HTTP, SOAP, WSDL, and many other Web technologies. XQuery is based on XML in the same way that an object-oriented language is based on objects, or SQL is based on tables. Just as SQL queries tables to create tables, XQuery queries XML to create XML. Unlike SQL result sets, however, the XML output of an XQuery may be directly useful to a data consumer, for example, a SOAP message, a document suitable for publication, or a Web page.

XML is the only data structure in XQuery. XQuery is designed to simplify operations most commonly needed when querying XML to produce XML results. For instance, XML structures can be quite complex, so XQuery has path expressions specifically designed to be able to easily find items in any XML structure. In contrast, conventional languages parse an XML file and navigate the resulting structures using an XML API, which requires significantly more code. In XQuery, an

XML constructor looks like the XML that is to be created. In conventional languages, creating XML requires laborious navigation using an XML API, adding new nodes to a tree. XQuery supports joins using a FLWOR expression, similar to SQL's SELECT-FROM-WHERE, so data from one or more sources can easily be combined and restructured. Typically, FLWOR expressions use path expressions to locate items in one or more XML structures, doing whatever joins are appropriate, and use constructors to create new XML structures as output. The same actions in a conventional language would typically require seven to 20 times more code.

Many enterprises have multiple data sources and multiple data consumers.

### **XQuery Simplifies Data Integration**

SOA generally requires data from many sources. XQuery was designed for data integration, as well as XML processing, and can easily do joins on data from different sources. From the beginning, the language was designed to be able to query anything that can be represented as XML, whether physically stored in XML or viewed as XML via middleware. To the query writer, all data looks like XML. Some of this data is physical XML, including conven-

tional XML documents and SOAP messages. Some of this data is converted on-the-fly to XML using adaptors.

For instance, if an XQuery vendor provides an adaptor for EDI messages, the message is converted to XML when it is queried. Some of this data is never converted to XML, but merely viewed as XML in the XQuery environment. For instance, an XQuery implementation that supports relational data generates SQL from an XQuery, queries the database in SQL, and converts the SQL result sets to XML, which is returned to the XQuery environment. In XQuery, all these different data sources can be treated as XML, making it easy to query multiple data sources, combine their data, and create XML structures containing the result.

When choosing an XQuery implementation, make sure that it fits into your computing environment and can handle the data sources needed in your architecture. The XQuery implementations from most database vendors are designed to query only data stored in their database, but most companies have more than one database, and data not found in a database. The XQuery implementations from application server vendors or XML integration server vendors can query a wider range of data sources, but require the adop-

tion of their server. If you are writing Web services in the Java environment, make sure your implementation supports the XQuery for Java API, which is a JDBC-like interface for XQuery. Also, the performance of XQuery implementations varies dramatically. Make sure that you test performance, especially if you are using XQuery for relational data or very large XML files.

### Defining Data Services in XQuery

In many SOA architectures, a data access layer that returns data as XML without granting direct access to the database and data integration layers is helpful. This approach protects access to the original data sources, frees applications from the details of data access, and ensures that changes are reflected consistently in all applications. The layer should support data services defined at the business logic level.

A data service can often be implemented using an XQuery and its parameters. For instance, an XQuery that creates a portfolio might have an external variable that identifies the user that would be bound for a particular user at run-time. Frequently used queries are prepared to improve performance. If the data access layer lets clients use HTTP to invoke XQueries

and provide parameters, the same data service can be used by a variety of clients.

### Conclusion

When designing a service-oriented architecture, enterprises should give careful attention to their data. Key criteria for success include the ability to efficiently process XML, the format used in all requests and responses, and the ability to easily incorporate data from a variety of sources, processing them together with data from incoming requests to create responses. XQuery can dramatically simplify service-oriented architectures, increasing programmer productivity and improving performance. XQuery can be used to define general data services that can be used throughout the enterprise. Using XQuery for XML processing, data integration, and data services will simplify your programs, increase programmer productivity, and improve the performance of your systems.

*Jonathan Robie is XML program manager at DataDirect Technologies and a co-inventor of XQuery.*

*Brian Anderson is director of product strategy at DataDirect Technologies. [www.datadirect.com](http://www.datadirect.com)*