November 2005

Managing XML for Maximum Return

By C. M. Saracco
IBM Software Group

CONTENTS

1. Overview 2
2. Why XML? 2
3. Managing XML: Need, Benefits 3
4. Managing XML: Options 4
4.1 Large Objects and Tables 5
4.2 Decomposition 5
4.3 XML-Only Data Management 6
4.4 Hybrid Data Management 7
5. Managing XML: IBM Solution 8
5.1 Relational Extensions 8
5.2 Native XML Support 9
5.3 Early Successes 12
6. Summary 13
7. References 14

1. Overview

In an industry rife with acronyms, one three-letter combination makes many information technology leaders shudder: ROI (return on investment). Perhaps that's because ROI for any given project – or any given investment in supporting infrastructure – is usually difficult to quantify or predict. Yet few technology initiatives are funded without a convincing business case that describes the anticipated business value.

This paper explores the need for – and value of – managing XML data. It also reviews key technology alternatives and outlines which options may be most appropriate based on your business needs. But first, it addresses perhaps the most obvious question: why should you care?

2. Why XML?

Since its debut in the 1990s, XML (eXtensible Markup Language) has emerged as a critical enabler to various technology initiatives. Service-oriented architectures (SOA), enterprise application integration (EAI), enterprise information integration (EII), Web services, and standardization efforts in many industries all rely on or make use of XML as an underlying technology.

Why? XML provides a neutral, flexible way of exchanging data among different devices, systems, and applications. Data is maintained in a self-describing format to accommodate a variety of ever-evolving business needs. Free software is available to help firms create, process, and transform XML data. All major industry vendors provide some level of XML support in their software offerings, and many have sought to exert considerable influence over XML-related standards – a sure sign of the technology's importance. Indeed, few industry analysts question the importance of XML today, and some are quite bullish on its prospects. ZapThink, for example, projects that the market for XML information exchange will approach nearly $3.8 billion by the end of the decade.

The business drivers behind XML's popularity are straightforward:

- A demand for increased business agility and efficiency
- A need to contain costs and "do more with less"
- A mandate to conform to regulatory requirements or comply with *de facto* industry standards

Let's look briefly at each of these in turn.

Building an agile business that can quickly respond to new market demands and competitive pressures implies that the underlying IT infrastructure must be easy to adapt and evolve. For example, tracking new information about customer preferences or buying behaviors can't

translate into a significant overhaul of a firm's production database; such an undertaking would be too time-consuming and costly. Similarly, firms can't afford to have the success of a new business partnership or acquisition hampered by an inability to exchange information between different systems.

Cost containment implies a need to make maximum use of new and existing IT assets. It counters the notion of "rip-and-replace" inherent in some technology proposals. SOA enables firms to create building blocks – or services – for their IT assets, thereby promoting greater code reuse and a more adaptable infrastructure. XML is emerging as the preferred format for services to receive and publish data. It runs on a wide range of hardware devices, it's supported by proprietary and open source software, and it can accommodate a variety of data content.

Regulatory requirements and industry-specific initiatives are also driving the deployment of XML. As more business transactions are conducted through Web-based interfaces and electronic forms, government agencies and commercial enterprises bear greater responsibility for preserving the original order, request, claim, or submission. Doing so can be essential for legal reasons and good customer relations. Again, XML provides a straightforward means of capturing and maintaining the data associated with these electronic transactions; indeed, electronic forms are commonly based on XML. Furthermore, consortiums in many vertical industries and application-specific areas have already begun to define XML-based schemas to promote exchange of data. These include such diverse efforts as ACORD in the insurance industry, FpML and FIXML in the financial services industry, RosettaNet in supply chain management (SCM), XBRL for reporting in business reporting applications, and others.

Finally, many firms are revisiting their proprietary electronic data interchange (EDI) efforts in favor of XML-based solutions. Cost savings are part of the reason. According to one study published in *Computer Economics,* XML often supports business-to-business transactions more economically than EDI. Indeed, 88% of the surveyed XML users received a full return on their investments compared with only 65% of the EDI users. Furthermore, EDI solutions were more likely to exceed total cost of ownership (TCO) expectations than XML-based solutions. More than 40% of the EDI users suffered from higher-than-anticipated TCO, while only 17% of the XML users did so.

3. Managing XML: The Need and Benefits

All this is leading many organizations to search for a way to effectively manage their messages and documents written in XML. Often, their motivation is straightforward: as XML becomes more critical to the operations of an enterprise, it becomes an asset that needs to be

shared, searched, secured, and maintained. Depending on its use, XML data may also need to be updated, audited, and integrated with other data. File systems aren't well-suited to supporting many of these tasks, particularly when scalability, concurrency, recovery, transaction management, and usability issues are taken into consideration. Database management software is a more appropriate choice, although until recently support for XML in many commercial offerings was somewhat limited.

Benefits of storing – or persisting – XML in a database management system vary according to the specific system in use. In a moment, we'll discuss several common architectural options and the trade-offs among them. However, potential benefits include:

- Improved employee productivity
- Improved IT resource utilization
- Reduced labor costs
- Quicker "time to value" for certain applications

You'll learn how this is possible as we explore different options for managing XML data and review IBM's solution.

4. Managing XML: The Options

The growing use of XML hasn't been lost on database management systems (DBMS) vendors. Relational DBMS vendors began extending their products to accommodate "unstructured" and "semi-structured" data years ago, while other vendors built new, specialized DBMS products specifically to support XML. More recently, some relational DBMS vendors, such as IBM, moved to merge these two distinct efforts into one offering. The result is a multi-functional DBMS that works efficiently with data modeled in both tabular and hierarchical structures. Because XML files typically consist of nested hierarchies, the ability to effectively store, search, and update data in these hierarchies is significant.

Early attempts to support "non-traditional" forms of data often involved straightforward extensions to commercial relational DBMS products, and some of these extensions were ultimately applied to managing XML data. For example, character and binary large objects (CLOBs and BLOBs) are two data types commonly used to store the entire contents of an XML file as a single column in a row of a table. Furthermore, some vendors enable users to "shred" or decompose an XML document across multiple columns in one or more tables.

These early efforts to extend relational DBMS products to accommodate various forms of "non-traditional" data had merit and can be successfully used to address the needs of certain XML-based applications. However, each of these technologies introduced limitations that ultimately led

some vendors to offer "native" support for XML data, which we'll discuss shortly.

4.1 Large Objects and Tables

Character or binary large objects (CLOBs or BLOBs) are one means of storing XML data in a tabular structure. By storing the XML document intact in a column of a row within a table, users don't need to break their document into pieces and map these pieces into various columns of one or more tables. Thus, the data modeling effort is simple and straightforward. Furthermore, complex joins aren't needed to reconstruct the original XML document because the document was never decomposed prior to storage.

However, using large objects – character or binary – has its drawbacks. Searching and retrieving a subset of the document can be expensive. New indexing technology may be needed to avoid the high cost of parsing XML for each query to determine which portions of the document satisfy the specified search criteria. Furthermore, updating large objects can be expensive. Often, client applications must provide the entire document to the DBMS for update, even if it only changed a small portion of it. This can result in unacceptably high processing costs, particularly if the XML document is very large.

4.2 Decomposition ("Shredding") into Tables

Performance problems for retrieving and updating portions of XML documents stored in large object columns led some vendors to offer document decomposition technology. This approach enables an administrator to map the elements and attributes of an XML document to columns in one or more tables. XML document values are then stored in these tables without their original tags.

"Shredding" XML data enables users to work with it in a purely tabular format, which implies several advantages. Users can leverage their existing SQL programming skills, as well as popular query and reporting tools, to work directly with selected portions of the "converted" XML data. This minimizes the need to develop new skills, which can translate into a higher level of productivity and even shorter application development cycles. Furthermore, runtime performance issues may be more predictable. No new indexing technology needs to be introduced, and updates against the converted XML data can be handled as efficiently as any other updates to data in standard SQL columns.

Unfortunately, the benefits of decomposing XML data often depend on the nature of the underlying XML document. This is because many XML documents contain heavily nested parent/child relationships and irregular structures. Shredding such documents may require a large number of tables, some of which may need to have values generated for foreign

keys to capture the relationships inherent in the original XML documents. As an example, one firm with 1500 electronic forms needed more than 30,000 tables to contain their data.

Even in simpler cases, the contents of a single electronic form can seldom be normalized into a single table. Thus, mapping the XML data to a relational schema and managing the resulting environment can result in considerable labor for a database administrator. In some cases, it may not even be practical to shred an XML document, not only because of its internal complexity but because it may have many sparse attributes (information that's "missing" or irrelevant for a given record). Modeling such documents using a normalized relational data model is often too complex and expensive; however, de-normalizing the model may not be feasible because of built-in database limits for the maximum widths of rows or the maximum number of columns per table.

Moreover, querying a "shredded" document can require complex SQL statements that include many joins. Such statements are often difficult to develop and tune; this increases development costs, impedes "time to value," and ultimately causes runtime performance problems that impact the productivity of multiple users.

Furthermore, changes to the XML schema often break the mapping between the original XML schema and the relational database schema, resulting in added labor costs. For example, the introduction of multiple email addresses for a single customer may require that a new table be created to comply with good database design principles (particularly normalization). However, introducing this new table implies implementing new primary/foreign key constraints, granting appropriate user access privileges, changing certain application logic, etc. Such work can be substantial. Consider a relatively optimistic case in which a firm might need to update its 1000 electronic forms once or twice a year, and each form was mapped to a mere three tables. This would result in 3000 to 6000 schema changes that would have to be manually managed.

Finally, any inherent ordering of elements or any digital signatures associated with the original XML document are lost when the document is decomposed into columns spanning one or more tables. For some applications, preserving the original form of the XML document – along with any digital signature – is critical.

4.3 XML-Only Data Management

Technical challenges associated with managing XML data in commercial relational DBMSs as large objects or through decomposition services prompted several firms to build XML-only database management products from scratch. By storing data in a hierarchical format and supporting a query language designed explicitly for XML data (XQuery),

these products avoided many of the performance, schema management, and usability problems associated with other approaches.

However, this new breed of XML-only DBMS offerings failed to garner significant customer interest or support. Industry analysts estimate that the combined revenues of all XML-only DBMS products represent a tiny fraction of overall DBMS sales. Indeed, several early entrants into the XML DBMS field have gone out of business, shifted their focus, or been acquired by other firms.

The reasons why XML-only DBMS products have struggled vary. Many firms are reluctant to introduce a new, unproven DBMS environment into their IT infrastructures. Integration with existing relational DBMS products may be limited or non-existent, which poses a problem for the many firms that need a cohesive, enterprise-wide view of their critical data assets. Support for high levels of scalability, reliability, and availability are seldom robust in new DBMS products. Finally, few database administrators and application programmers have substantial skills in managing XML databases or querying collections of XML data using XQuery. Thus, introducing a new, unproven DBMS into an IT infrastructure can compromise its efficiency. Although skilled XML programmers may enjoy some productivity gains, high integration costs and system management challenges often mitigate overall benefits.

4.4 Hybrid Data Management

The growing use of XML and the lack of a comprehensive, efficient solution for managing this data along with other forms of corporate data has led to the development of hybrid database management software. Such software seeks to provide first-class support for both tabular and XML data structures, as well as SQL and XQuery. The objective of such systems is to preserve the benefits associated with commercial relational DBMS offerings – including high levels of scalability, reliability, availability, concurrency, and customer support – while making it easy to manage and integrate existing corporate data with data modeled in hierarchical XML structures.

Achieving this objective is best accomplished by building on a proven relational DBMS base and crafting new core capabilities within the system to efficiently index, search, and store XML data. Ideally, such a DBMS should optionally support validating XML data prior to storage and provide a simple means of coping with changing schemas.

For firms with existing relational DBMS environments, this approach enables them to derive new value from their investment. A hybrid DBMS enables users to seamless share, store, retrieve, and update both existing corporate data and XML data that had previously existed only in flat files or transient messages. Furthermore, it minimizes the amount of new skills required to incorporate XML data into their database

environments, reducing labor costs and potentially speeding project delivery cycles.

For firms concerned only with XML data management, the hybrid approach offers them a reliable and scalable infrastructure, the ability to leverage "relational" tools against their native XML data through the use of SQL/XML functions, the option of searching data in a query language designed for XML (XQuery), and the backing of major industry vendors. In addition, labor-intensive tasks such as mapping XML schemas to relational schemas and writing complex SQL statements simply to query "converted" XML data are minimized or eliminated, which can improve both staff productivity and development cycle time.

5.0 Managing XML: The IBM Solution

IBM's solution for managing XML data provides customers with a highly flexible, reliable, and efficient DBMS environment. Firms can use large objects or decomposition technology to model their XML data in tables, just as they've been able to do for years. However, DB2's new Viper release allows users to store, query, and process XML data in its hierarchical structure without sacrificing traditional DBMS support for transaction management, security, query optimization, and the like. We'll briefly review the first two options and then focus on the new capabilities in DB2's Viper release for managing XML data in its native format.

5.1 Relational Extensions for XML

For years, DB2 customers have been able to use large objects, user-defined types, user-defined functions, and administrative tools to store XML data within tables. IBM continues to support these options and has even provided new capabilities its Viper release. Collectively, IBM's relational extensions for managing XML data with large objects or through XML decomposition and publishing technologies enable users to:

- Store XML data in a single column with minimal DBMS awareness of the internal structure of the XML data.
- Extract elements from an XML document and store their contents in multiple columns of one or more tables, in effect "converting" the XML data to tabular data. With Viper, new shredding technology supports larger XML documents and offers potential performance improvements.
- Store XML data type definitions (DTDs) for use in validating XML data.

- Compose and publish XML fragments from tables through the use of SQL/XML functions and mapping files. For example, users can write SQL queries that return results as well-formed XML.

- Invoke system-supplied stored procedures to administer their databases, generate XML documents, or "shred" XML documents into tables.

- Invoke system-supplied functions to insert XML documents, retrieve XML documents, extract element content or attribute values, and update XML documents.

These capabilities are most useful for situations in which users primarily want to perceive XML data as being part of a tabular data model that can be queried using SQL or SQL/XML functions. Applications that require high performance for searching and retrieving subsets of XML documents, that must cope with frequently changing XML schemas, or that require extensive use of XQuery and navigational expressions may be better served through new DB2 capabilities for native storage and management of XML data.

5.2 Native Storage and Management of XML

DB2's Viper release, now in beta, features extensive new support for XML as a first-class data type. This "native" support for XML includes new storage techniques for efficient management of hierarchical structures inherent in XML documents, new indexing technology to speed retrieval of subsets of XML documents, new capabilities for validating XML data and managing changing XML schemas, new query language support (including native support for XQuery as well as new SQL/XML enhancements), new query optimization techniques, integration with popular application programming interfaces (APIs), and extensions to popular database utilities. The result is a single DBMS platform that offers the benefits of a commercial relational environment and a native XML database environment.

Fig. 1 illustrates the overall architecture of DB2 Viper. Both tabular and hierarchical storage models are supported through common engine components. Furthermore, client applications that need to work with both traditional SQL and XML data can use either SQL/XML or XQuery statements (or a combination of both). Full support for DB2 transaction semantics, security mechanisms, and distributed computing constructs (such as stored procedures) are supported for both SQL and XML data.

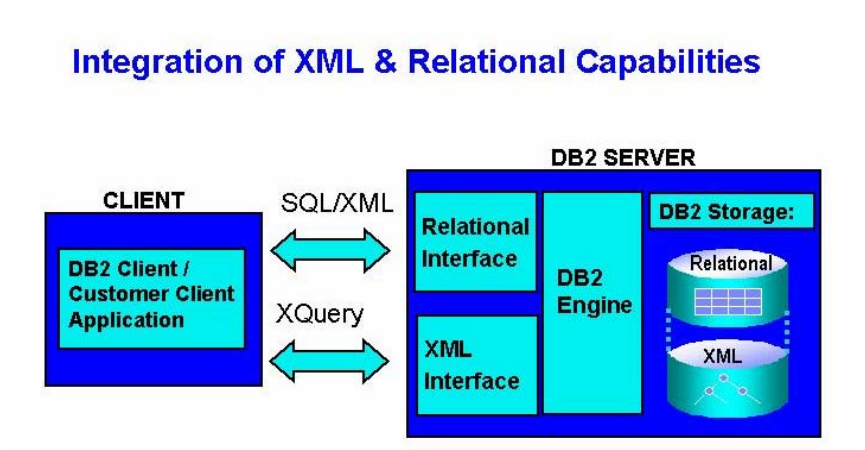

Fig. 1: DB2 Viper architecture

With DB2 Viper, collections of XML documents are captured in tables that contain one or more columns based on a new XML data type. This enables users to employ familiar SQL data definition language (DDL) statements to create database objects for storing their XML, although DB2 will treat the XML data different internally. Specifically, it will automatically employ a custom storage management architecture that preserves the hierarchical structure of the original XML data and supports rapid retrieval of such data (or portions of it).

Tables created with XML data types may also contain columns with "traditional" SQL data types, including numeric data, character strings, date/time data, and others. Here's a simple example of how to define a table that maintains both types of data:

CREATE TABLE mytable (msgID INT PRIMARY KEY NOT NULL, msg XML)

After creating tables, users can issue INSERT statements or invoke the DB2 IMPORT facility to add data to their tables; such data may include both "traditional" SQL data types as well as DB2's new XML data type. Inserting or importing data in this manner makes it easy for customers to leverage their existing DB2 skills. However, these mechanisms hide the fact that DB2 manages the XML data in a way that's quite different from how it manages traditional SQL data types. In short, a parsed representation of each XML document is stored in its hierarchical format. If users instruct DB2 to validate their XML data prior to storage based on an XML schema, DB2 will annotate all nodes in the XML hierarchy with information about the schema types; otherwise, it will annotate the nodes with default type information. Furthermore, DB2 will automatically split portions of XML documents across multiple database pages as needed.

To help speed searches, users can create indexes for specific elements or attributes of their XML documents. Such indexes are defined over XML patterns – essentially, XPath expressions without predicates – and can speed the retrieval of queries targeting specific portions of XML documents. Full text search over XML documents is also supported, and specialized text indexes can be defined to improve performance of such searches.

Because DB2 Viper is a bilingual product, users can search both XML data and traditional SQL data types with SQL or XQuery. Indeed, a single query can span XML and traditional SQL data stored within a single database. Furthermore, with WebSphere Information Integrator, firms can even write SQL-based queries that join and union XML data maintained in external files with data in DB2 and other non-IBM data sources. While details of supported query language capabilities are beyond the scope of this paper, it's important to note that IBM's implementation is based on existing and rapidly emerging standards for both SQL and XQuery.

To efficiently process queries of XML data, DB2 leverages cost-based query optimization technology to evaluate different data access strategies and select a low-cost option. The large size of many XML documents, the complexity of query predicates found in many XPath expressions, and the need to preserve the order of elements contained within XML documents prompted IBM to develop new query operators and a new join algorithm specifically to speed searches of XML data. The join algorithm provides for concurrent evaluation of "and" and "or" query predicates, as well as employs multiple cursors on XML indexes to locate the desired information quickly. Use of this new join technology is transparent to application programmers; DB2 will automatically evaluate queries and determine when it's beneficial to use it.

To serve a wide range of programmer needs, DB2 Viper enables native XML data to be accessed through Java (JDBC), C (embedded SQL and call-level interface), COBOL (embedded SQL), PHP, and Microsoft's .NET environment (through the DB2.NET provider). To help administrators monitor and tune their databases, familiar facilities such as DB2 Snapshot, RUNSTATS, and EXPLAIN provide a "snapshot" of database activities at a given point in time, collect statistics about the nature of data within a database (including XML data), and report on the access path selected by the query optimizer (including new information about the use of indexes on XML data). Furthermore, DB2's built-in repository stores information relevant for validating XML data (including XML schemas and data type definitions) as well as other XML artifacts.

5.3 Early Successes

Interest in DB2's new XML support has been strong, with firms in various industries already evaluating an early release of the technology. Storebrand, one of Norway's largest providers of insurance and financial services, is among these firms. Storebrand was an early adopter of SOA and Web services, perceiving these technologies as important for helping the firm improve its customer focus and deliver greater value at a lower cost. According to Storebrand Senior Enterprise Architect Thore Thomassen, the firm considers XML to be an important integration mechanism. As such, XML data is an important asset that must be stored, managed, shared, and analyzed to support various business initiatives.

Thomassen noted that the firm's early experiments with the alpha release of DB2 Viper were promising. Although Storebrand found each of DB2's three storage options – large objects, decomposition (or shredding), and native support – to be useful, Thomassen discovered distinct advantages for native XML support in certain comparative test situations. For example, preliminary work in test environments indicated that DB2's native XML support could help them

- Reduce the time it took to generate an internal report from more than 1 day to less than 10 minutes.
- Cut the I/O portions of select Web services an average of 65% and decrease the maintenance time for these Web services by 20%.
- Implement a schema change (in response to new business requirements) in a few minutes instead of requiring a full day to prototype and test the change and a full week to deliver it.
- Minimize the labor required to program six new database search and retrieval scenarios. Using native XML features, a programmer completed the task in a ½ hour. By contrast, the same work required 2 hours with decomposition and 8 hours with CLOBs.

In particular, Thomassen reviewed the results of one test project and noted, "Development time using the (DB2) XML native store is overall radically improved over shredding. Also, shredding often results in complex mappings, which mean that the developer needs deep competence in constructing SQL."

Indeed, internal IBM studies have also demonstrated similar potential benefits. One comparative study published on IBM developerWorks involved a Web-based PHP application that used XML for capturing customer input and publishing data. Specifically, it explored the design and coding requirements for storing and searching the application's data using a traditional relational database environment (in which XML is

shredded prior to storage) and DB2's native XML support. Storing the XML data in its native format simplified the database schema considerably, resulting in only three tables of two columns each instead of four tables with up to nine columns each. Furthermore, certain aspects of the application, such as populating the database, were written with only one-third of the code. Finally, accommodating a new user requirement that resulted in an XML schema change was a more straightforward undertaking because administrators didn't need to change the underlying database schema and application programmers didn't need to rewrite the logic of their code.

6.0 Summary

XML messages and documents have emerged as key assets in many organizations, forcing IT executives and architects to find an effective means of managing this data for maximum advantage. Previous technology initiatives often fell short of achieving this goal. However, the new IBM DB2 Viper release features hybrid database management technology that incorporates proven relational capabilities with first-class support for storing, searching, sharing, validating, and managing XML data. The result is a reliable, scalable platform that provides high performance for accessing and integrating "traditional" corporate data as well as XML data.

Early adopters are already noting the labor savings, shortened development cycles, and improved flexibility that DB2's XML support offers. In today's environment, such benefits can quickly translate into key competitive advantages.

7. References and Related Reading

7.1 DB2 Materials

Beyer, Kevin, et. al. "System RX: One Part Relational, One Part XML," *SIGMOD,* June 2005.

Bhambhri, Anjul. "Firing up the Hybrid Engine," *DB2 Magazine,* Quarter 3, 2005. (http://www.db2mag.com or http://www.software.ibm.com/data/db2/xml)

"Comparing XML and relational storage: A best practices guide," *IBM White Paper,* IBM Corp., March 2005, IBM order number GC34-2497-00. (http://www.software.ibm.com/data/db2/xml)

"The IBM approach to unified XML/relational databases," *IBM White Paper,* IBM Corp., March 2005, IBM order number GC34-2496-00. (http://www.software.ibm.com/data/db2/xml)

Nicola, Mattias and Bert Van der Linden. "Native XML Support in DB2 Universal Database," *Proceedings of the 31st Annual VLDB,* 2005. (http://www.vldb2005.org/program/paper/thu/p1164-nicola.pdf)

Singh, Hardeep and Amir Malik. "Use DB2 native XML with PHP," IBM developerWorks, Oct. 21, 2005. (http://www.ibm.com/developerworks)

Wong, Cindy. "Overview of DB2's XML Capabilities: An introduction to SQL/XML functions in DB2 UDB and the DB2 XML Extender," IBM developerWorks, Nov. 20, 2003. (http://www.ibm.com/developerworks)

7.2 General Materials

Bourret, Ronald. "XML and Databases," self-published at www.rpbourret.com, September 2005.

Linthicum, David S. "The ROI of Your SOA," *ebizq.net,* July 10, 2005.

Schmeizer, Ronald. "The ROI of SOA," *ZapFlash,* Jan. 27, 2005, Document ID: ZAPFLASH-20050127.

"XML Economics Beat EDI in B2B Transactions," *Computer Economics,* July 2004.

7.3 Acknowledgments

Thanks to Seeling Cheung, Matthias Nicola, Hardeep Singh, and Thore Thomassen for their contributions to this paper.

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

11-05