

Integrating XML and Relational Database Technologies: A Position Paper

By

Giovanni Guardalben
Vice President of Research and Development
HiT Software, Inc.
gianni@hitsw.com

and

Shaku Atre
President and CEO
Atre Group, Inc.
shaku@atre.com

Revised 8/12/04

Abstract

The XML markup language should be the *lingua franca* of data interchange, but its rate of acceptance has been limited by an underlying mismatch between XML and legacy relational databases. This, in turn, has created a need for mapping tools that can make it relatively easy to save XML documents on a legacy RDBMS. Such tools should help XML fulfill its potential and gain wide acceptance.

XML is an ideal tool for data interchange. XML documents can be saved on either XML native databases or legacy relational databases. XML data is not only exchanged, but also processed and stored. As a result, the issue of storing XML data effectively and efficiently becomes paramount.

In spite of the need for efficient XML storage, native XML databases still make up only a tiny percentage of all installed databases. Why? Mainly because existing relational databases are critical infrastructure in most organizations. However, storing XML data meaningfully into relational tables is anything but straightforward because the XML and the relational approaches are built on different principles. The intrinsic differences in the two approaches create a need for tools that can map XML to relational data.

1 Introduction

XML is an open standard for defining data elements on a Web page and business documents. In contrast to the HTML markup language, which defines how elements on a Web page are displayed, XML defines the structured information those elements contain.

XML was originally developed as an application profile of SGML to use over the Internet. But the ease of both writing applications that process XML document and creating XML documents has made XML an instant success for a variety of other application domains, too. In fact, a truly amazing number of applications based on XML are currently under development. One area where the benefits of XML have become immediately apparent is data exchange. For instance, XML has been combined with EDI (Electronic Data Interchange).

There are clear benefits to using XML. First, the design of XML is formal and concise, so it's relatively easy to write data-exchange protocol processors that rely on standard XML components. Second, since XML documents are human-readable, a developer can figure out what the content means by simply inspecting the XML document. Third, momentum is building to create standardized XML protocols—also referred to as XML Schemas—for almost any type of business. So there's an excellent chance that in the near future a majority of transactions will be carried out using XML as the underlying infrastructure protocol.

The shift from binary protocols to XML-based protocols resembles the computing industry's move from mainframe-centric terminals and central processing units to the client/server paradigm of distributed computing. Like client/server before it, XML requires new ways of storing data. In the shift from mainframe-centric to client/server computing, a major side effect was the development of relational database servers. Similarly, in the case of the move to XML protocols, there is a noticeable trend toward not only exchanging XML data, but also storing it directly as XML data. To support this practice, some relational database vendors and independent middleware vendors have released native XML database products and extended relational database features to support XML.

Before discussing the integration of XML and relational database technologies, let's look briefly at native XML database technology. In many instances, native XML database technology seems derived from the object-oriented database technology of the early '90s—especially in its indexing techniques. Compared with object-database technology and its lackluster track record, native XML databases benefit from an application's need to retrieve and store data directly as XML data. While this is a tremendous advantage for XML data that is content or structured text rather than data, it becomes cumbersome when the same XML content has to be mapped to corporate data, such as that stored in relational tables. Given the existing RDBMS installed base, unless the XML middleware industry can seamlessly integrate native-XML-database and

relational-database technologies, it will be extremely hard for the native-XML technology to thrive.

Looking ahead, we believe an increasing number of developers will shift to an XML-based development paradigm. But this shift will not bring about a revolution in the underlying database technology. On the contrary, we expect that relational technology will remain the dominant storage technology—albeit, with important extensions to XML—and that it will coexist with the new native-XML database technology. Of course, native-XML database technology arose due to the intrinsic differences between the relational-database paradigm and the XML paradigm, the so-called XML-relational impedance mismatch. But we also believe that just as the object-relational mapping technique simplifies object persistency on relational repositories, XML mapping technology can bridge the gap to properly retrieve XML data from relational databases and persistent XML data on relational databases.

1.1 XML-to-Relational Mapping

There are several important advantages to using XML-to-RDB mapping to integrate XML and relational data:

- The new XML paradigm of development based on standard software components—such as XML parser, filters and processors—can coexist with existing legacy (relational) data repositories without requiring a re-architecting of the database.
- XML developers need not be involved in database-development issues. That's because the relational data is presented as a virtual XML document—that is, an XML view of relational data—only basic skills in XML development are needed.
- Given the intrinsic differences between XML information-sets and relational data, automatic conversion tools cannot infer all the implicit semantics that exist between relational tables and fields. As a result, design-time mapping specifications provide the necessary flexibility required to explicitly define hidden data semantics—especially the containment semantics of hierarchical XML structures.

To achieve seamless integration between XML and relational data, the mapping technology must:

- Be based on a declarative approach. Mapping users should not be required to know how to implement mapping operations. Instead, that should be part of the functionality of the underlying mapping infrastructure. In this way, project maintenance is simplified because any project-design changes imply only mapping-definition changes rather than modifications to the application code.
- Rely on W3C Schema or DTD (Document Type Definition) instances to describe the format of the XML document that represents the XML view of the relational data. This is important both for compliance with standards (W3C Schema is a W3C Recommendation) and because W3C Schemas contain XML document

metadata information that describe classes, or templates, of XML documents. This is a better abstraction than establishing relationships between single instances of XML documents and relational data.

- Establish links between XML element/attribute definitions and table-field definitions as obtained from RDBMS Catalog queries. This way, mappings are relationships between XML metadata and RDBMS metadata. As a result, the abstraction layer is consistently the same from XML to relational data.
- Use XML documents to define mapping relationships. Besides being consistent with the overall XML technology, one can use standardized XML software components both to parse XML data and the XML mapping information.
- Be generated by GUI (graphical user interface) applications. Although mapping definitions are XML documents and can be edited by text editors, the XML structure of mapping definitions can be rather complex and hard to maintain. For complex mapping relationships, a mapper application is a must.
- Generate SQL statements automatically after being processed. Considering the complexity of mapping relational information to XML infosets, it is more efficient to let the mapping do the hard work of producing the suitable SQL statement required to generate the relational data to map.
- Be independent of the relational database management system. The mapping processor should rely on standard database-access middleware such JDBC (Java DataBase Connectivity), ODBC (Open DataBase Connectivity), or OLEDB (OLE DataBase). It should also reduce the need for support of vendor-specific features.
- Integrate easily with existing XML processor tools such as XPATH (XML Path Language), XSLT (EXTensible Stylesheet Language Transformation) and the upcoming XQuery language.

1.2 Features of the XML-to-Relational Mapping Infrastructure

To be effective, XML-to-RDBMS mapping tools should include several important features, including mapping definitions, tools to support mapping operations, and the ability to integrate query, selection, and transformation XML processors. Let's examine each in detail.

1.2.1 Mapping Definitions

Some features of XML mapping definitions should be direct references from the requirement list. In other words, mapping definitions should rely on the W3C Schema or the DTD standards. They also should link element- and attribute-definitions to fields from the database table and column catalog datasets. To take advantage of XML's flexibility, the syntax for the mapping definitions should itself be based on XML.

The choice of specific keywords that make up the model for describing mapping definitions should be made carefully. Although no standards are currently being

developed for mapping definitions, we recommend either of the following two approaches:

- Define an XML syntax derived from the W3C Schema syntax by annotating it with mapping extensions. This is known as the element/attribute annotation approach.
- Define an XML syntax derived from the W3C Schema syntax by extending it with new schema types. This is the type-refinement approach.

The first approach, the W3C Schema to be mapped has extra elements either as annotations, using the standard W3C Schema construct, or as new mapping elements, a proprietary approach by means of additional namespaces.

Under the second approach, new types are derived from the ones defined in the schema to map. These refined types carry the extra information necessary to establish the mapping rules.

Currently, the W3C Schema standard supports both approaches. Until a new standard comes along, the choice between the two is rather subjective.

What to include in mapping definitions? Here are some of the important pieces that should be present:

- Links between element/attribute definitions in the schema definition and table fields in the database catalog column datasets.
- Database referential integrity constraints to allow the proper retrieval and storage of complex hierarchical information.
- Optional database-table information: column data types, precision, scale, etc. Other relevant pieces of information include primary key information, flags for auto-incremental fields and other corollary database information. The purpose of this extra data is to import XML data into a database, even in the presence of missing tables.
- Complex mapping support. This includes declarations for embedding scripting information in mapping definitions and XML syntax for mapping SQL expression on XML elements or XML expression on SQL fields.
- Predicate support. Sometimes it's useful to specify constraints on the size of the XML record set, even at map design-time. This usually happens when problem requirements are fixed. Therefore it's convenient to support a syntax for adding predicate support, both when retrieving XML data and when storing XML data. A further extension to this mechanism is support for parametrical predicates.
- XML aggregation support. When retrieving XML data from relational data, it's difficult to combine hierarchical information in such a way that parent elements group more than one child element together—that is, when many child elements share the same parent element. Typically, this happens as a result of SQL join queries on the database server.
- Optional information. This includes connectivity details if the mapping is to be done only for specific DBMS servers, locations of the schema definitions to map

(URI vs. stream vs. files), and project-wide options to support specific customizations.

1.2.2 Tools to Support Mapping Operations

As described earlier, mapping definitions contain all the semantics of the mapping process. As mapping technology improves, there will be a growing number of modeling changes, and producing and maintaining the syntax of mapping definitions will become so complex that manual editing will become the exception, not the rule. Although the physical protocol of the mapping definitions is preferably XML-based, automated tools—GUI front-end applications—should be used to generate and edit mapping instructions.

Generally speaking, there is no unique model for producing mapping information. But many software-mapping tools are now available, and most employ strikingly similar usage patterns. Most, in fact, contain at least two tree-views—or list-views, if the data-structure layout is flat—of the information to be mapped. These tools also include facilities to link single elements from one view to the next. Common approaches are drag-and-drop, link activation dialog boxes, and variations of cut-and-paste techniques. In addition, mapped tree elements are tied to one another by way of graphical links.

In the case of XML-to-relational mapping tools, one tree-view should present a basic tree representation of a W3C Schema or DTD document. Another tree-view should provide a reduced representation of a DBMS catalog tree: qualifiers/databases or owners/tables/columns. The leaf nodes of one tree-view—XML elements and attributes on one side, table fields on the other—are the ones to connect to the leaf nodes of the other tree-view.

If a tool supports advanced features such XML or SQL expressions, it also needs full-fledged expression editors to reduce the likelihood of generating syntactically invalid expressions. Also, if scripting support is offered scripting editors should be available. Scripting debuggers can also help reduce the probability of generating run-time scripting bugs. Finally, mappers should also provide support lists; this eliminates the need to manually type hard-to-remember database and XML identifier names.

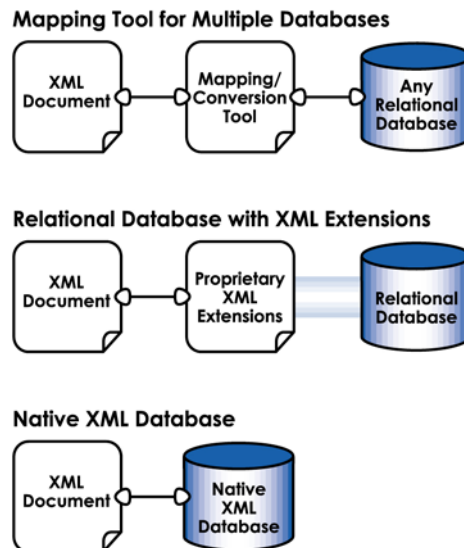
1.2.3 Integration of Query, Selection and Transformation XML Processors

The mapping process described here is essentially a methodology to present subsets of relational databases as XML views. The larger objective is to switch from a database-centric approach of application development to one that is based entirely on the XML paradigm. One side effect of this paradigm shift is the possibility of using standardized XML processors—including XQuery, XPath and XSLT—to further query, filter, and transform the XML views obtained from relational data by means of mapping techniques.

One obvious difficulty of running XML processors on mapped XML views is that XML views are not materialized unless cached in memory, stream, or file. This may become a blocking issue if the XML view obtained from the mapping process is large. Either the cached view is too large for memory or disc space, or the time needed to post-process the XML view by an XML processor is unacceptably long.

To overcome these problems, any of several strategies can be used. Here are two options currently used by commercial products:

- Develop mapping-aware XML processors. Such processors understand mapping definitions and translate XML processor statements into native SQL statements. Then the same SQL statement is merged with the one generated by the mapping processor. While this is the most ad-hoc solution, it is also the most labor-intensive. That's because XML processors are complex software components that need to be maintained whenever the standards are updated.
- Use standard XML processors from software vendors or open source. If a developer uses off-the-shelf XML processors to perform post-processing operations after the mapping has already taken place, they face the problems examined previously. So their objective is to first reduce the size of the XML record set generated by the mapping process, and then post-process it by means of third-party XML processors. To reduce the size of the XML record set, they can perform an initial parsing of the XML processor query statement to extract only the relevant predicate information (that is, SQL where conditions) to append to the SQL generated by the mapping process.



2 Major software vendors' XML-to-relational integration products

Several of the largest software vendors -including IBM, Microsoft and Oracle- currently offer products for XML-to-relational data integration. The recently released Microsoft's ADO.NET framework also provides a generalized option for relational data access via XML. Finally, HiT Software Allora addresses integration from a platform agnostic, declarative approach. Let's take a look at these products and their relative strengths and weaknesses.

2.1 Oracle and XML

Since 2001, Oracle has developed several modules to easily exchange XML with its leading RDBMS. The Oracle XML Developer's Kit is a set of components, tools and utilities that eases the task of building and deploying XML-enabled applications whereas Oracle XML DB allows organizations to store and search XML documents in a similar way to other XML native databases.

2.1.1 General Information

Oracle has released various versions of its XDK—XML Developer's Kit—since Oracle8i. With Oracle XDK 10g, the entire set of XML libraries and components—for Java, C, and C++—that were previously released separately have now been packaged and included as a single XML Developer's Kit with corresponding components and libraries.

Oracle XDK contains XML Parsers, support for XMLType, full internationalization support, XSLT processors (XSLT 2.0) and XML utilities with the basic building blocks for reading, manipulating, transforming and viewing XML documents. The Java, C and C++ kits also include an XML Schema processor. The Oracle XDK 10g now supports the Java Architectural Binding for XML (JAXB) specification that provides also a Java object access to XML documents. These XML support tools are basic and do not provide functionality aimed at simplifying XML and database integration. In fact, it is only in the Java and the PL/SQL Developer's Kits that we find utilities directly addressing the XML-to-DBMS integration problem. More specifically, in the Java Developer's Kit, we find the XML SQL Utility (XSU) and the XSQL Servlet Utility. Similarly, in the PL/SQL Developer's Kit we find the XML SQL Utility for PL/SQL.

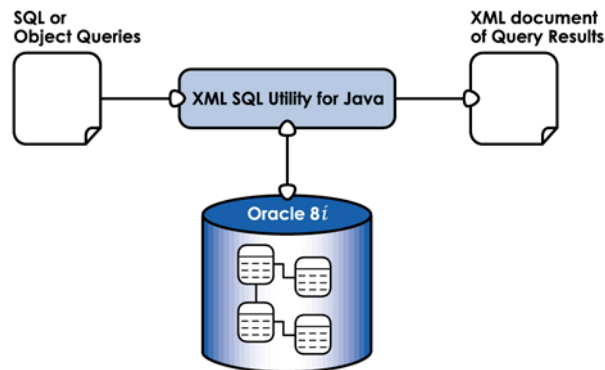
2.1.2 The XML SQL Utility (XSU)

Let's analyze the XML SQL Utility for both the Java and the PL/SQL development environments. According to Oracle, the XML-SQL Utility (XSU) enables one to:

- Transform data retrieved from object-relational database tables or views into XML.
- Extract data from a XML document and, using a canonical mapping, insert the data into the appropriate columns/attributes of a table or a view.

- Extract data from a XML document, and apply this data to updating or deleting values of the appropriate columns/attributes.
Note: in order to insert one XML document into multiple tables/views, you need to create an object-relational view over the target schema. Finally, if the view is not updatable, one can get around this using instead-of insert triggers.

This utility can work with any JDBC drivers but is optimized for Oracle's JDBC drivers. Oracle, does not make any guarantees or provide support for XSU running against non-Oracle databases.



Here's how the XSU utility accomplishes its objectives:

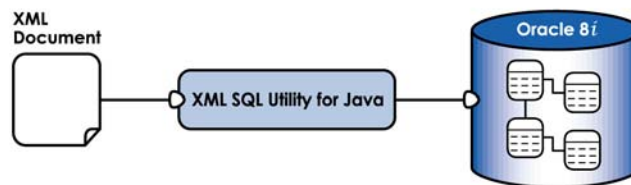
- XSU comes with two modes of operations: the XSU Command Line and the XSU Java (or PL/SQL) API. In the former, the command line invokes the Java class OracleXML. In the latter, Java applications rely on the OracleXMLQuery and the OracleXMLSave Java classes.
- When using the command line, you can either obtain an XML document from a DBMS (by calling the `-getXML` option) or insert an XML document into a DBMS by calling the `-putXML` option.
- When using the Java API, the developer has increased flexibility and can generate an XML document or an XML DOM info set from a query. Also, when the result set is large, it can be paginated into manageable chunks.
- When going from XML to SQL the XML attributes are ignored; thus, there is really no mapping of XML attributes to SQL.
- When using the Java API, the developer can either execute an SQL Select query or call an Oracle Stored Procedure that returns a result set.
- When either retrieving or storing XML, some column-to-XML element/attribute mapping can be used to improve the quality and organization of the resulting XML or XML import. Here's how mapping is supported:
 - Default SQL Mapping: In the generated XML, all rows are enclosed in a Rowset tag.
 - Default SQL Mapping: The Rowset element contains one or more Row elements. Each of these contains one or more elements whose name and

content are those of the database columns and values of the retrieved tables.

- o Object Mapping: When the developer wishes to generate an XML document that represents complex relationships between tables, object-relational SQL syntax is provided—that is, the CreateType object-relational SQL statement. Using these SQL extensions, the developer can generate complex containment operations by specifying hierarchies of objects and sub-objects. An object-relational view which maps to the desired XML document structure is created over the database schema.
- o Object Mapping: Some forms of customizations are possible on the resulting XML. When storing XML using objects, the input XML must match the standard XML format of Oracle object types.

If the XML document doesn't perfectly map into the target database schema, there are two things you can do about it:

- Modify the Target. Create an object-relational view over the target schema, and make the view the new target.
- Modify the XML Document. Use XSLT to transform the XML document. The XSLT can be registered with the XSU so that the incoming XML is automatically transformed, before any mapping attempts are made. Note that this solution does not offer the best performance.



2.1.3 Oracle XML DB

Oracle XML DB is neither a separate product nor a separate option that you must install. Oracle XML DB refers to the collection of XML features and technologies built directly into Oracle9i Database. A key feature is the XML DB Repository. This repository enables you to store XML documents directly in Oracle Database. Once your XML documents are in the repository, you can access your XML data in either an XML-centric or a relational-centric manner.

To store XML data in your database, you simply write an XML document file using FTP, HTTP, or WebDAV—all industry-standard protocols. Getting XML data out of your database can be as simple as executing a SQL query or reading a file using one of those same protocols.

It is also possible to access XML data in the repository by going straight to the underlying table. You can write queries directly against this table, but those queries must be XML-aware. To facilitate access to XML data from reporting tools designed for use with

relational data, you can create a view that lists the correspondence between the database fields and the XML nodes using XPath. The view and index permit the efficient execution of standard relational queries on XML documents. Oracle XML DB is usually better suited for document-centric applications.

2.1.4 Comments on the Oracle XML tools

The Oracle XML-SQL Utility (XSU) illustrates Oracle's commitment to XML-enabling its database offerings. Oracle is also extending the database engine to act more like a native XML database (Oracle XML/DB). Notable features of Oracle's kit are the breadth of its XML support and, thanks to Oracle's size and commitment to standards, its guarantee to continuously maintain and update the XML product offering.

However, based on our understanding of Oracle XML Developer's Kit, there are still areas of RDB-XML support that are not yet covered, particularly for more complex and XML schema-centric integration projects:

- Limited mapping support: The default SQL mapping provided by XSU maps XML elements to table or query column names and values. Though the developer can rename columns, or produce hierarchical XML documents using object-relational views, it is a tedious, programmatic process for complex projects. There are in fact a couple of challenges to this solution. First, to XML-enable a DBMS, the DBMS itself has to change, and it is difficult to create complex relationships between XML elements/attributes and SQL expressions. When a developer needs to map an existing Oracle database to a complex schema, the best alternative is first to create an XML representation of the relational database with a fixed schema and then apply XSLT. This two-step solution has the disadvantage of being error-prone, difficult to maintain and it may not perform well.
- Lack of graphical development tools: To simplify the generation and storage of complex XML documents, it's often desirable to have graphical tools that can help the developer define mapping between SQL structures and XML elements or attributes. But Oracle provides only API or command-language tools.
- Oracle provides excellent solutions to create XML from the database according a fixed-schema and to create a new database design from a XML-schema. However, mapping a complex industry XML schema like UCCNET, ACORD, IFX to an existing Oracle database stays problematic as the developer needs to design complex object-relational models or add an additional XSLT processing step.

2.2 IBM DB2 XML Extender

2.2.1 General Information

IBM's DB2 XML Extender is a set of administration and programming tools designed to handle XML integration with IBM's DB2 database management system. In contrast to

Oracle's tools, IBM's product does not rely on different kits for specific programming languages. In fact, the XML Extender programming tools are DB2-stored procedures and can be called from any DB2 SQL client software, whether standard or proprietary.

IBM offers two basic types of DB2-to-XML integration modes: XML Column and XML Collection. XML Column is recommended if you need to store and maintain the document structure as it currently is. Typically, this is used for XML content management. The other mode, XML Collection, is recommended when SQL data is either decomposed from incoming XML or assembled to produce outgoing XML documents. When data is moved to DB2, it is managed and updated as regular SQL data. Given that we are interested only in XML data integration with databases, we'll concentrate only on the XML Collection's functionality.

2.2.2 Administration Tools

IBM's documentation, a book entitled *XML Extender Administration and Programming Version 7*, states that the IBM XML Extender for DB2 consists of administration tools and programming tools.

There are two basic administration tools: the XML Extender administration wizard and the *dxxadm* command. Both rely on specific administration stored procedures. The administration wizard is a Java application. The command-line application runs on multiple platforms.

Both administration tools rely on the presence of a special file called the DAD (Database Action Diagram) file. The DAD file is used to map the structure of the XML document to the DB2 tables from which you either compose or decompose the XML document. In a DAD file root, element, text and attribute nodes can be mapped to table column names or query column names. That is, you can map either a table or a query statement. When mapping a query statement (an activity known as SQL mapping), only marshaling is allowed. When mapping from one or more tables (known as RDB mapping), one can also apply conditions, which get translated into *where* conditions. Data types must be provided for each mapping entry. When using RDB-mapping, SQL statements are created by the XML Extender.

A DAD can be created by using either a text or XML editor. Another option is to use the administration wizard, though its support for creating the DAD file is only slightly more powerful than that of a regular editor.

DAD files can be registered in the DB2 database. When doing so, a database becomes enabled. Enabled databases import and export XML efficiently because they don't need to parse the DAD file each time.

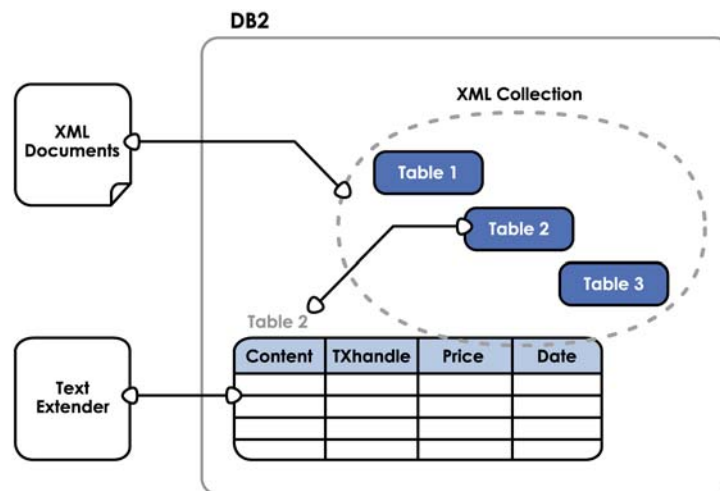
To pre-validate XML documents, DTDs (Document Type Definitions) can be imported into DB2. Validation is performed by the XML Extender before any other XML operation if the DTD is registered to a database.

2.2.3 Programming Tools

IBM's XML Extender programming tools provide two basic functionalities: XML document decomposition, or unmarshaling, and XML document composition, or marshaling.

The first functionality, XML document decomposition, supports two stored procedures: one for decomposing XML documents to databases that are not enabled (dxxShredXML), and another for decomposing to databases that are enabled (dxxInsertXML). The only difference between the two is the DAD file content that is required for the dxxShredXML.

The second functionality, XML document composition, is supported similarly: dxxGenXML is used for applications that do occasional XML generations, while dxxRetrieveXML is used for applications that make regular XML updates. Occasionally, DAD instructions can be dynamically modified by an override parameter that temporarily modifies the content of the DAD file.



2.2.4 Comments on the IBM DB2 XML Extender

Compared with Oracle's XML Toolkit, IBM's DB2 XML Extender supports a more declarative approach by means of the DAD configuration file. Basically, IBM's solution endorses the idea of having a design-time mapping session and run-time phase. We believe this approach is more convenient than the programmatic approach for two reasons: One, it clearly demarcates the modeling phase from the execution one, and two, it introduces an element of flexibility since XML modeling changes do not require changes in the application code.

However, IBM's DB2 XML Extender also suffers from a few limitations:

- It's not an XML-oriented development environment. Currently, developers who are dealing with XML documents deal exclusively with SQL middleware, because XML support is provided by DB2-stored procedures. While introducing XML to a project involves important programming shifts, such as viewing any data as XML instances, IBM's XML Extender maintains a SQL-centric view of development. Based on this approach, the XML API is of little use except to import and export files. Also, developers have to be knowledgeable both in the SQL and the XML programming environments. We believe this to be an important architectural limitation.
- It's available only for IBM DB2. Since the engine of XML Extender is available only as DB2 Stored Procedures, this architecture for XML-to-DB integration cannot support databases other than DB2. This turns out to be a severe limitation for XML applications and projects that need to be portable to many DBMS setups.

2.3 Microsoft SQL Server 2000 XML Support

2.3.1 General Information

Microsoft's objective is to make SQL Server 2000 a fully fledged, XML-enabled database server. With the new XML features introduced with SQL Server 2000 support, developers can use the Microsoft product to:

- Create XML Views using annotated XDR schemas.
- Use XPath to query XML Views.
- Retrieve and write XML data by means of XML extension to Microsoft Transact SQL.

The first two items provide support for abstracting SQL Server as a fully functional XML database server using XML queries rather than SQL queries as the prime query tools. The third item is a set of convenient XML extensions to the SQL language, intended for traditional SQL developers. Let's look at these Microsoft features in more depth.

2.3.2 Creating XML Views Using Annotated XDR Schemas

XDR schema syntax is Microsoft's proprietary format for describing XML schemas. XDR schemas provide a richer set of data types and other features than does the DTD standard, much like the W3C XML schemas. In fact, the next release of SQL Server will most likely replace these features with the W3C XML Schema Standard (XSD). Queries to the database, when used with annotations to describe the mapping of XML elements and attributes to the database tables and fields, return the results in the form of XML documents, or XML views. In this way, a developer can specify an XPath query against the XML view created by the XDR schema.

Microsoft XML views support a large set of annotations. A developer may not specify any annotation, thereby using a default mapping. The resulting XML is similar to Oracle and IBM's default mapping support.

By means of annotations, a developer can specify the tables to map (**sql:relation** annotation), the fields to map (**sql:field** annotation) and the referential integrity relationships (**sql:relationship** annotation). Static (that is, at mapping time) support for predicates—namely, the equivalent to SQL where conditions—is also available (**sql:limitfield** and **sql:limitvalue** annotations). Also, one can specify which XML elements or attributes are mapped to key fields in the SQL database, to enforce unity, etc.

Finally, XML Views can only be used to retrieve data (XML marshaling), and are read-only tools.

2.3.3 XPath Query Support for SQL Server 2000

The W3C XPath standard can be used as a standard query language for XML. The supported XPath for querying SQL Server is a subset of the full XPath standard because XML views generated by SQL Server are 'virtual' XML documents and are not materialized in a stream or file. There are, however, five important limitations:

- It cannot execute root query (/). Queries must begin at a top level <ElementType> in the schema.
- When an error occurs, an empty XML document is returned.
- XML views do not have a pre-defined order. Consequently, predicates that use document order are not supported.
- There are some limitations on the data types supported.
- Queries that require recursive predicate, and that need to use aliases, are not supported.

2.3.4 Retrieving and Writing XML Data

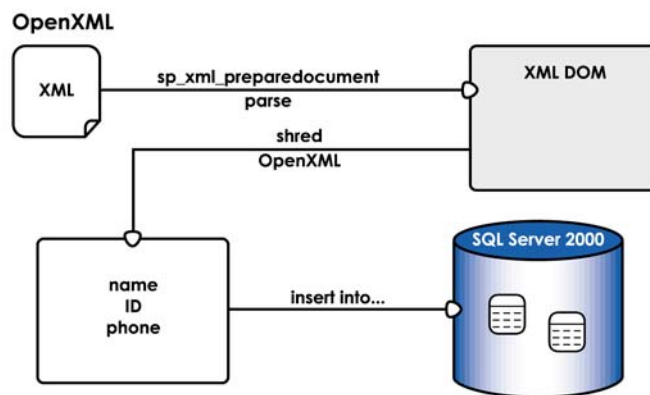
There are two basic methods for using extended SQL to operate on XML data. The first relies on an ad-hoc clause (FOR XML clause) for the SELECT statement. The second is the Transact-SQL OPENXML function to insert data formatted as an XML document.

The FOR XML clause of the SELECT statement is used to return results as XML documents rather than standard relational data. One can use any of three XML modes:

- RAW: This mode transforms relational row sets into XML elements with the generic identifier row.
- AUTO: This mode returns query results as nested XML elements. Table fields are mapped as XML attributes (and their names are strictly the field names), and table names are mapped as XML elements.
- EXPLICIT: This mode requires the query writer to specify the shape and names of the resulting XML document. Of the three modes, this one provides the most flexibility.

The Transact-SQL OPENXML keyword provides a row set over an XML document. As such, an OPENXML is similar to a table or a view provider. Using the OPENXML function, a developer can move an XML document to database tables. An OPENXML can be used in SELECT or SELECT INTO statements. To write queries against an XML document using OPENXML, you must first call **sp_xml_preparedocument**, which parses the XML document and returns a handle to the parsed document that is ready for consumption.

The OPENXML identifier is provided with a number of options to properly unmarshal XML documents into database tables. Among them is the ability to choose between attribute-centric mapping and element-centric mapping.



2.3.5 Comments on the Microsoft SQL Server 2000 XML Support

SQL Server XML Support illustrates Microsoft's high-level commitment to support the new XML standards. XML is now tightly integrated in the engine processors of SQL Server by means of SQL extensions and powerful DBMS integrated XML processors, such as XPATH. In the area of database and XML integration, Microsoft is not just adding XML features to its SQL Server product, but is also adding XML support to its new database access infrastructure, called ADO.NET (see next section for details).

Given that Microsoft has created such a feature-rich set of infrastructures, it's difficult to find any limitations. In fact, there are just two:

- Microsoft SQL Server 2000-Only Support: Since the overall XML support for Microsoft SQL Server is based on proprietary extensions, either by annotated XDR or SQL extensions – FOR XML clause or OPENXML identifier, it cannot be extended to other DBMSs unless those data sources are embedded within SQL Server as heterogeneous linked servers.
- Limited Mapping Support: Because SQL Server XML Support uses XDR annotations, it provides flexible declarative-mapping definitions. Column names can be mapped to existing XML elements and attributes, and XML names can be totally unrelated to database column names. Also, predicates and referential integrity can be input using annotations. So far, so good. But

Microsoft does not provide instructions on what to do when database tables lack either unique indexes or primary keys, or on how to create tables on-the-fly during a decomposition operation, to name just two of the important functionalities of a mapping tool that are missing.

2.4 Microsoft ADO.NET XML Support

2.4.1 General Information

Microsoft's ADO.NET (ADO still stands for ActiveX Data Objects) is a set of .NET data-access classes that form an integral part of the .NET Framework. ADO.NET is designed to provide data access for disconnected n-Tier applications. XML plays a crucial role as the data-representation language for ADO.NET data. It provides seamless access to data from data sources and XML documents. The ADO.NET object model comprises two families of objects: the .NET data-provider objects; and disconnected objects centered on the DataSet class object, which includes DataTables, DataRows, DataColumn, and Constraints.

In the former group, .NET data-provider objects, we have objects for accessing SQL Server, OLE DB data sources, and ODBC data sources. These objects are referred to as .NET data provider objects. They handle standard data-access and manipulation tasks for relational databases, including running stored procedures. The second group, based on disconnected objects, provides a collection of objects for data manipulation of any data sources, relational or otherwise. The DataSet objects operate independently of the actual connection to the database. Focused solely on data manipulation, they have no direct knowledge of the connection information of the data source.

DataSet objects are linked to the data-source connection by the DataAdapter or the DataReader objects. By splitting the data-source access and manipulation activities, Microsoft achieves a high level of flexibility. Also, the DataSet object includes referential integrity and many other features usually found only in an in-memory relational database. Another of Microsoft's major design decisions was to leave the task of supporting data to XML integration to only DataSet objects. In this way, ADO.NET is made to interoperate with other systems.

How do the major features of ADO.NET stack up from the point of view of XML-to-relational data integration? Let's start by describing the general functionality of a data provider. Then, we'll look into a DataSet object. Lastly, we'll discover how XML can be marshaled and unmarshaled to and from a data set.

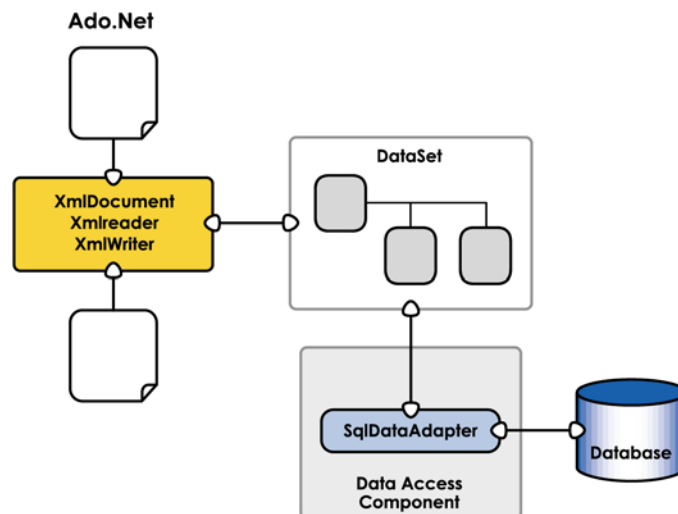
2.4.2 ADO.NET Data Providers

Compared with the data-access components of previous standards, such as ODBC drivers and OLE DB providers, .NET data providers are managed providers. That is, they

execute within .Net Framework CLR. These .Net Data Providers are used for connecting to a data source, executing a command, and retrieving data. Data providers can also handle basic data manipulation, such as updates and inserts.

These objects make up managed providers:

- Connection: Used to set a connection to a data source.
- Command: Once a connection is established, execute commands and return result sets using the Command object.
- DataReader: A fast, read-only, forward-only data stream aimed at providing quick access to data.
- DataAdapter: The basic link for exchanging data between a data source and data set. They can move any kind of data between a data source and DataSet object, as long as it is supported by a .Net provider.



2.4.3 The DataSet Object

This is the important second layer of the ADO.NET object model. The DataSet object is populated using a DataAdapter object. DataSets can be modified, and the changes can be committed to the data source using the DataAdapter Update method. The DataSet is a collection of fine-grained objects: Table objects, Relationship objects and Constraint objects (for data integrity). It is a virtual cache of a database. It includes data and schemas, relations and constraints.

Integration with XML is achieved either directly from the DataSet object or by using an intermediate class called the XMLDataDocument. The latter method provides extra flexibility because the resulting XML document is returned directly in DOM format. When

using DataSet WriteXML or GetXML, XML streams are read directly into the DataSet to populate it. You can also read/write XML Schema information by using special options.

2.4.4 Comments on Microsoft ADO.NET XML Support

ADO.NET infrastructure leverages Microsoft's 10-year experience with standardized data access frameworks, starting with ODBC, then OLEDB and ADO, and now with ADO.NET. If the future is anything like the past, ADO.NET will succeed and become a de facto standard for database access.

In terms of XML integration, ADO.NET was built from the ground-up with XML in mind. Nonetheless, a few aspects of ADO.NET are still wanting:

- **Microsoft .NET Platform Only:** As the name implies, the ADO.NET classes can be run only as managed code. This means the .NET platform must be available on the computer on which they're executed. Currently, this platform is primarily available for the Windows operating system.
- **Limited Mapping Support:** Mapping for the ADO.NET is achieved by associating mapping objects—specifically, DataTableMapping and DataColumnMapping objects—to DataAdapter objects. These mapping objects intervene when the Fill method of the DataAdapter object is invoked. These mapping objects link table names to elements and column names to elements or attributes. No provisions are made for expression, scripts, row combinations, and key identifications.
- **Programmatical-Only Support:** When mapping complex transformations between DBMS tables/query and XML, it is desirable to both rely on mapping declarations and use a generic mapping processor to execute the mapping declarations. By using a declarative approach—and more effectively, by relying on a GUI-based mapping tool—any changes to either the mapping or the XML Schema can be processed more easily than it would be by having to modify the ADO.NET application.

2.5 HiT Software Allora

2.5.1 General Information

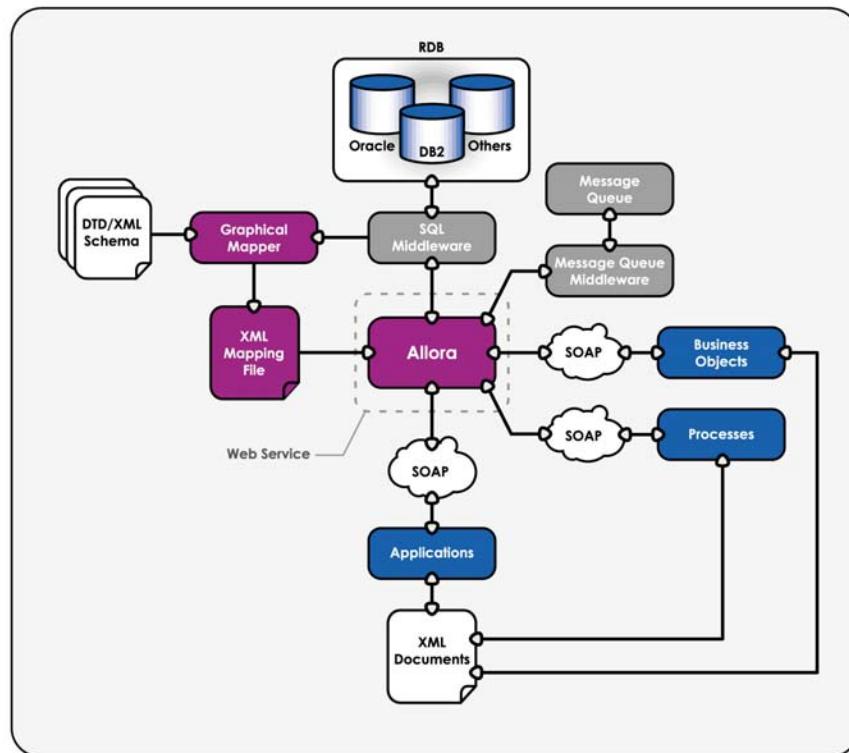
HiT Software's Allora is a XML-to-RDBMS integration middleware. It natively supports Sun's Java platform and in contrast to the other products described in this position paper, Allora is not a server environment, but rather a set of tools that can be integrated with server applications. HiT Software's Allora aims to solve the problem of integrating any relational data source to any XML schema document. Allora features two major software components: the Allora Mapper and the Allora Engine. It also includes several corollary tools, including wizards for major IDE applications like JDeveloper or Eclipse, adapters for different server applications, and Web service integration modules.

2.5.2 Allora Mapper

HIT's Allora Mapper is a Java swing GUI application that supports the creation of mapping definitions between a W3C Schema or DTD instances and table fields from a relational database. Link setting is performed using the drag-and-drop approach. To reduce the syntactical complexity of XML Schemas and DTDs, Allora Mapper represents these in a simplified, tree-view manner. At the same time, its field definitions show all relevant information from the database catalog, such as data type, precision, and scale.

Allora Mapper supports the creation and editing of links between element/attributes and table fields. It also supports the retrieval and establishment of referential integrity constraints. These are used to perform meaningful joins among different relational tables. Other major functions include support for the creation of predicates (which are eventually translated to the SQL Where clauses), as well as for SQL and XML expressions and dynamic parameters in predicate definitions.

One major feature supported by Allora is scripting. When design-time mapping designers use scripting, they have the freedom to operate on either SQL data (marshaling) or XML data (unmarshaling). The supported scripting languages are JavaScript and Jython.



2.5.3 Allora Engine

HiT's Allora Engine is where XML integration happens. It's a set of Java classes that take the mapping definitions from the Allora Mapper and then process them to perform XML export or import. The Java classes support an almost identical invocation model, syntactical differences notwithstanding.

According to this API model, XML operations are performed either in batch mode (with both full-fledged marshaling or unmarshaling supported) or record-by-record. When updating the database, isolation levels can be used for transactional support.

2.5.4 Other Tools

HiT's Allora product offers wizards to integrate Allora with IDE environments. Wizards are offered for Borland JBuilder, Oracle JDeveloper, IBM Eclipse or Sun Java Studio. These wizards generate source code based on the mapping definitions provided by the Allora Mapper. They also support Java Data Binding by means of "setter and getter" methods that simplify both the access and modification of XML record values.

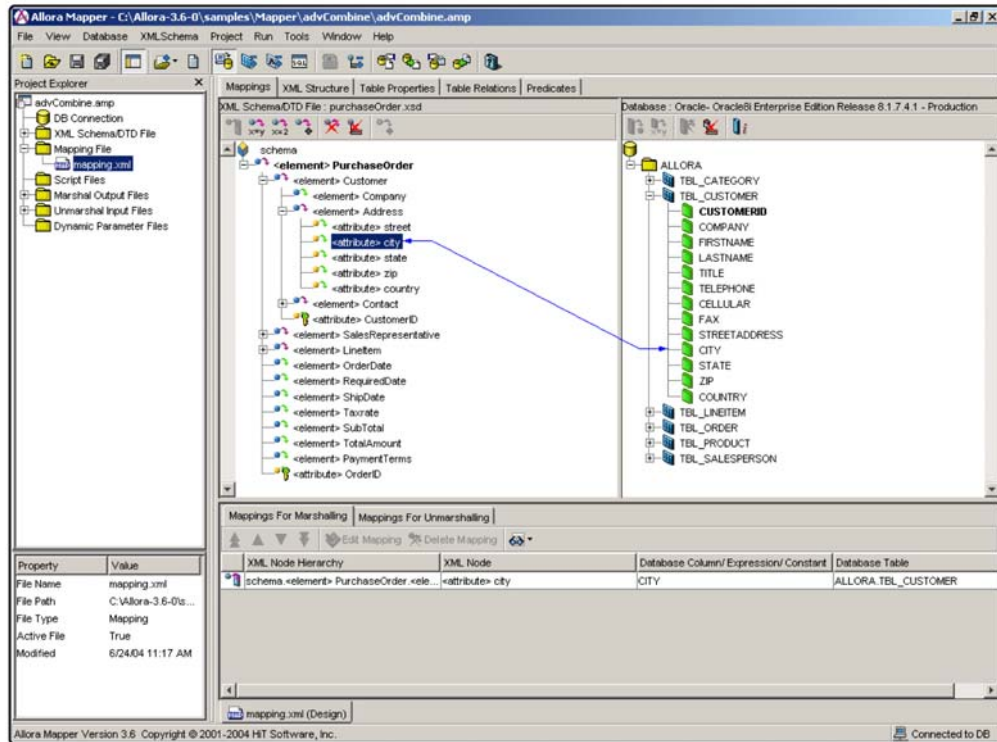
2.5.5 Comments on the Allora Product

Allora leverages HiT Software's more than eight years of experience in designing and implementing standards-based data access middleware, including ODBC, OLE DB, and JDBC drivers. With Allora, HiT Software has produced what could become a standard way to integrate XML with relational databases, regardless of the database brand. In fact, Allora is compatible with all JDBC and ODBC drivers currently available.

Here are Allora's most important features:

- Ability to handle any XML schema or DTD.
- A powerful GUI Mapper that simplifies the creation of mapping definitions.
- A declarative approach—namely, mapping definitions—rather than a merely procedural approach for mapping support.
- A set of SOAP (Simple Object Access Protocol) interfaces to its Web service engine for building distributed applications.

Looking ahead, an important technological challenge for Allora will be to integrate mapping technology with XML processors, including XQuery, XPath and XSLT. By providing seamless and efficient integration, Allora will combine the benefits of customized mapping with the flexibility of generalized and standardized XML processors.



3 Conclusions

XML is here to stay. With the noteworthy exception of HTML, no other standard has been so successful so quickly. One critical measure of XML's success is how pervasive it has become in the computing industry. It's safe to say that virtually every branch of the computing industry has embraced the XML standard in one form or another. The database industry is no exception, and in the area of database management systems, XML has a major role to play.

While the industry generally agrees that XML and relational data will become interchangeable, converting XML to and from relational data is often difficult. In fact, when a project requires that relational data be converted to or from complex XML formats (XML schemas or DTDs), there is no silver bullet. One offered solution, automatic conversion utilities, fails to address the intricacies of the required conversions. Instead, complex-mapping instructions must be drawn to specify the conversion methods. To aid this process, a GUI application can prevent syntax errors and maintain semantic consistencies among all the tabular data in relational databases. Also, standard XML processors such as XQuery, XPath and XSLT should be integrated with the mapping processor to provide the maximum amount of query and transformation flexibility on the database data.

No vendor is currently covering all aspects of XML integration, but significant progress is being made on an almost daily basis. This opens some opportunities for middleware companies that are focusing on specific customer needs or are providing solutions that apply across all database vendors and address higher levels of mapping complexity

Abbreviations and Acronyms Used in This Document

API: Application Program Interface
DAD: Database Action Diagram
DBMS: Data Base Management System
DTD: Document Type Definition
GUI: Graphical User Interface
HTML: HyperText Markup Language
JDBC: Java Data Base Connectivity
ODBC: Open Data Base Connectivity
OLEDB: OLE Data Base
RDBMS: Relational Data Base Management System
SGML: Standard Generalized Markup Language
SOAP: Simple Object Access Protocol
SQL: Structured Query Language
URI: Uniform Resource Identifier
W3C: World Wide Web Consortium
XML: EXTensible Markup Language
XPath: XML Path Language
XSLT: EXTensible Stylesheet Language Transformation
XSU: XML-SQL Utility

Product and Organization References

HIT Software/Allora: <http://www.hitsw.com>
IBM/DB2 XML Extender: <http://www-3.ibm.com/software/data/db2/extenders/xmlxt/index.html>
Microsoft/ADO.NET: <http://msdn.microsoft.com/vstudio/techinfo/articles/upgrade/adoplusdefault.asp>
Microsoft/Universal DataAccess: <http://www.microsoft.com/data/>
Oracle/ XML SQL Utility: <http://www.oracle.com/technology//tech/xml/index.html>
Sun Microsystems/JDBC: <http://java.sun.com/products/jdbc/>
XML Schema. <http://www.w3.org/xml/schema/>
SGML: ISO 8879: 1986, <http://www.iso.ch/iso/en/ISOOnline.frontpage>
XML: <http://www.w3.org/XML/>
XPath: <http://www.w3.org/tr/xpath>
XQuery: <http://www.w3.org/XML/Query>
W3C: <http://www.w3.org/>

~~~~~  
**For More Information about HiT Software and its Allora solutions:**

HiT Software Inc.  
4020 Moorpark Ave., Suite 100  
San Jose, CA 95117  
**Tel:** (408) 345-4001  
**Fax:** (408)345-4899  
**e-mail:** [info@hitsw.com](mailto:info@hitsw.com)  
**Web:** [www.hitsw.com](http://www.hitsw.com)

~~~~~  
For More Information:

Atre Group, Inc.
303 Potrero St., Ste. 29-303
Santa Cruz, CA 95060
831-460-9300 (v)
831-460-9400 (f)
www.atre.com
info@atre.com