Server-Side XML:
Taming the Tower of Babel

1.0 Overview

XML appears to be all things to all people and is being touted as the panacea for everything that is wrong with the Web and application development. The hype surrounding XML is well deserved, but determining how to maximize the practical benefits can be dizzying. To understand the different roles it can play, you need to consider XML for all types of data and across all layers of an application.

Most of the attention XML has received focuses on client-side document management, but where XML can make the most dramatic impact on application development is when it is applied to the broader problem of all application data, and on the server side. The server side is where data consumers and data providers all come together - each talking a different language. The result is a middle tier that begins to look like a "Tower of Babel." XML – the universal format for data interchange – can tame this chaos by acting as the object model for distributed data. Server-side XML can solve one of the most expensive problems of developing enterprise applications: data integration in the middle tier.

Integration and data sharing in the middle tier is elegantly addressed by XML's universality and versatility, which present unique management requirements. An appropriate data management system conveniently and efficiently delivers XML data to the applications for simpler development and end-user performance. Object Design's ObjectStore® database, with its native storage of XML data objects and its unique ability to distribute across the middle tier, is ideally suited to manage server-side XML and be the underlying technology for a data integration solution – one that Object Design is committed to building.

2.0 Why XML?

XML is up to the task of being the universal data interchange format because it is simple to read and self-describing. XML is fully portable because it is ASCII and can be transferred using standard protocols such as HTTP. Once received, any application can interpret the XML by using a simple parser and derive information by navigating the tree. Each field and attribute is delimited with an identifying tag, so every data element comes with identifying metadata. In addition, the nesting of the fields imposes structure on the elements, so the relationship between data elements can be inferred.

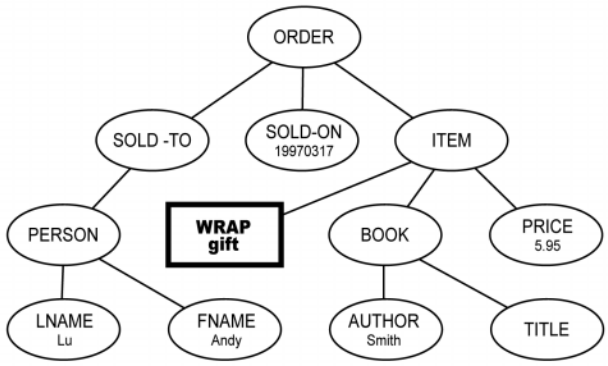
In the simple example in Figure 1, XML is used dynamically to represent an e-commerce transaction. The data is easy to understand, and updates are handled easily. The tree-like structure is flexible enough to manage the richly structured data objects of e-commerce as well as stock trading, manufacturing, and other data-intensive vertical industries. Because of the tags and the structure, an application can parse and interpret any XML data without prior knowledge of the specific format.

XML data can further describe itself by including a document type definition (DTD). The DTD defines the structure of the data object up front so that applications can ascertain what attributes are included and what fields are optional. This enables application servers to interpret data on the fly and respond dynamically. Because XML is self-describing, each data structure does not require custom code – applications can be written generically to universally interpret and handle data regardless of the source or the exact structure.

An important aspect of XML is its extensibility. When data is shared between systems, the schema is often updated to record, for example, new item or inventory information. XML can absorb new fields without disrupting the existing data structure, so database schemas do not need to be redesigned and applications do not need to be rewritten. The example in Figure 1 shows how an e-commerce data object is easily extended to include a new item.

Figure 1 An XML Example – an e-commerce transaction with update

```
<ORDER>
<SOLD-TO>
   <PERSON>
     <LNAME>Lu</LNAME>
     <FNAME>Andy</FNAME>
   </PERSON>
</SOLD-TO>
<SOLD-ON>19970317</SOLD-ON>
<ITEM>
   <PRICE>5.95</PRICE>
   <BOOK>
     <TITLE>Pi</TITLE>
     <AUTHOR>Smith</AUTHOR>
   </BOOK>
</ITEM>
<ITEM>
     <WRAP>gift</WRAP>
</ITEM>
</ORDER>
```



These unique properties of XML make it an ideal format for the universal interchange of data. Not only is it flexible to handle richly structured data without complex object modeling, but it also supports dynamic data which has never had a practical solution. XML enables applications servers to be open to freely share and manipulate data.

2.0 XML: Universal and Ubiquitous

Because of its popularity, XML is being considered as the solution to a wide range of problems and determining where it can make the most impact takes some investigation. To fully understand the benefits of XML requires looking at the entire XML market and differentiating between the various solution sets. While the benefits of client-side XML are most immediately apparent and have received considerable publicity, XML on the server side can make the greatest impact on application development. After all, the benefits to the client can only be realized if there is full support from the server.

2.1 XML for Document Data and Application Data

Most of the publicity that XML has received pertains to its ability to store document data - i.e. human readable data. Widespread usage of the Web and the well-understood problems associated with searching for information across the Internet have earned XML the label: “HTML on steroids.” Where HTML delivers formatting information, XML delivers data, which separates content from presentation in Web pages and enables better searching. Using XML to enhance Web pages has broad reach for its ability to easily deliver semantic meaning with each page.

XML is also being used today for more comprehensive document data such as product manuals or engine part descriptions. This is inherited from SGML, XML’s more intricate ancestor. XML is successful at handling complex document data because while being easier to use, it is still able to handle the complexities of highly structured information. The hierarchical nature of XML gives document producers an easy way to combine text modules, and the tags facilitate targeted searching.

While XML is good for document data, XML is sophisticated and can be applied to the broader job of being a universal format for application data – i.e. any data that is consumed by a process – with far more motivating results. XML is flexible enough to be used for the exchange of any type of data - including e-commerce, network configuration and stock ticker information - or any data that is shared between application servers. Because XML is hierarchical, it is designed to represent hierarchically structured data as well as a C++ or Java class can, but without locking developers into a programming language. And also unlike C++, readable tags travel

with the data so application servers do not need custom code to parse and interpret shared data.

XML is a robust and effective format for application data and its neutrality makes it a common denominator for all applications. This qualifies XML as an interchange format that can be applied to any system integration or business-to-business project. With a universal data format, the development of communication code is simplified and proprietary modules can be replaced with general-purpose functions making application development faster and cheaper.

2.2 XML in the Client, Back End and Middle Tier

Just as XML's wide range of benefits can be applied to all types of data, it can also be used in all tiers of an application, bringing different advantages to each. Figure 2 shows the breadth of application areas for XML.

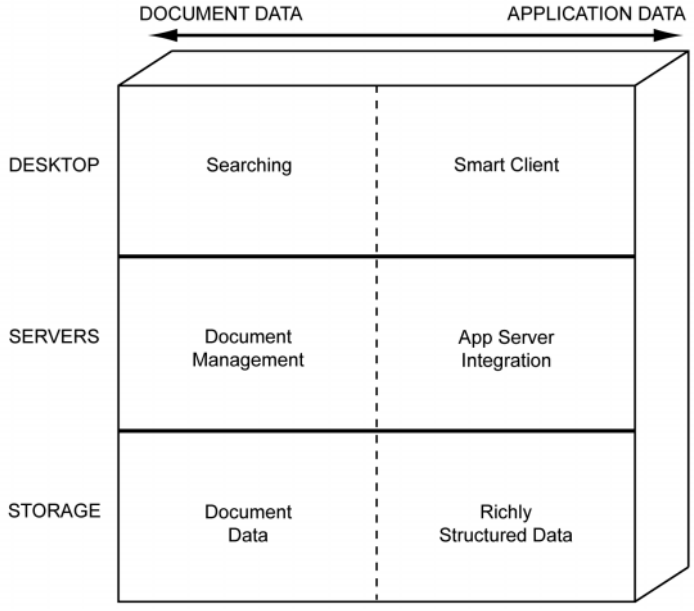
XML in the client makes data that is delivered to desktop application "smart data," allowing the applications to do more without help from the server. It can do more effective searching, querying and manipulation of the data and do more client-side processing. This makes the client a smart client that can search document data or process application data.

XML can also be used in the back end as an efficient storage format for any highly structured information - whether it is document or application data. If data is hierarchical when it is in use, it makes sense to preserve the structure in storage. Storing it in another format requires the conversion of the data and the inefficiency of imposing an inappropriate structure onto the data. In the case of a traditional relational database, richly structured data has to be decomposed into its elements, mapped to rows and columns, and then reconstructed when accessed. XML is a convenient and efficient storage format for any richly structured data.

In the middle tier, XML can be used to solve some of the most difficult issues of application design and system integration. The middle tier is where communication and data sharing takes place between application servers and back end storage engines. Because XML is self-describing, it can be used as a universal format for data interchange and simplifies the potential chaos in the middle tier. XML can be used to neutralize the data, reducing the effort required to build and support a middle tier.

Figure 2

The Benefits and Uses of XML



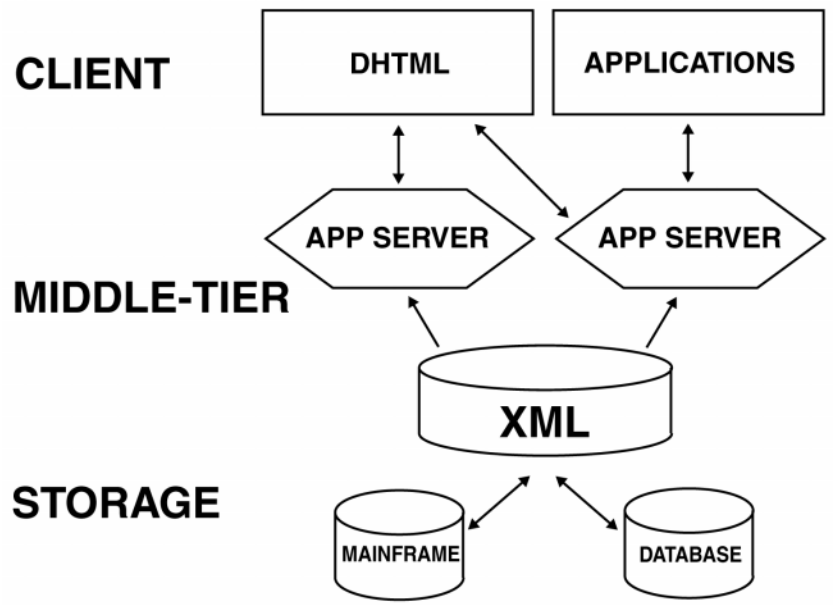
3.0 XML and Data Aggregation

One of the most compelling uses for XML is to integrate the data between application servers and data sources. This takes place in the middle tier where business logic and back-end systems meet, and often requires considerable custom code to glue the components together.

For example, an on-line catalog application needs to share data with a product database, an inventory system and perhaps a transaction provider. The traditional brute force approach to facilitating data sharing is to generate custom interchange code between each pair of data producers/consumers, which grows geometrically as the application includes more integrated servers. This code is non-value-add and is not reusable.

The alternative is to use XML as the universal interchange format, which allows application servers to look at all the data through a single, logical view. If XML is used as the common denominator, each integrated component only needs a single XML adapter and the code base is limited to linear growth. What makes this possible is that XML elements are labeled with self-describing tags, so data objects can be dynamically evaluated as they are received.

Figure 3 XML as a Distributed Data Model in Multi-Tier Applications



By using XML as a distributed data model, the middle tier turns from being a "Tower of Babel" into an efficient layer for data exchange. The challenge is to manage the XML data on the server side.

4.0 Server-Side XML

An integrated application – whether it is an enterprise system or an application intended to interoperate over the Internet – involves data-sharing between application servers and back end storage devices. The architecture is multitier, as is shown above in Figure 3, and the data often has a different format in every component. If XML is to be used as the common denominator for data exchange, it has to be managed on the server side to provide efficient access and manipulation of the data objects. The solution for data integration is server-side XML.

Server-side XML has to be managed to serve the needs of applications. The requirements of an XML management solution are:

- **Performance** – application servers must have high-speed access to the data and be able to execute rapid queries across large sets of aggregated data.
- **Extensibility** - applications treat data as dynamic objects and XML data management must be able to support schema changes on-the-fly.
- **Middle Tier Support** – XML data must be served to multiple application servers in the middle tier with performance and data coherency.

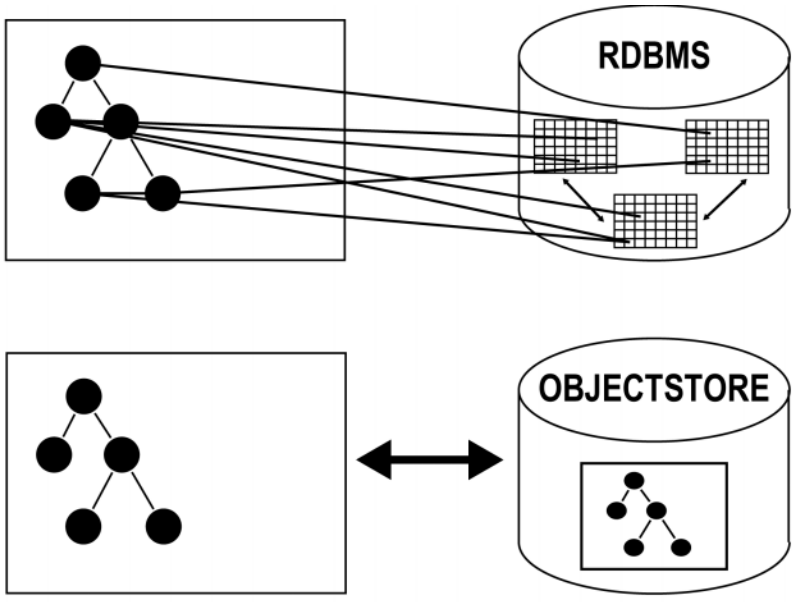
Managed, server-side XML allows application servers to look at aggregated data through a unified, logical view. As a result, application servers do not need to manage data translation, or work around the data models of the other components. This simplifies the development of application servers and separates the business logic from middle tier data integration issues. Managing server-side XML successfully, however, has some unique and fundamental requirements that merit special considerations.

4.1 Performance

A major requirement of server-side XML is a high-performance, distributed data cache in the middle tier. Since data from original sources is not necessarily in XML, a server-side datastore eliminates the need to construct and deconstruct the object at every access and make a round trip to the back-end storage device. A successful datastore is one that can store XML natively so data access and manipulation can be done efficiently.

Because the nested fields of XML are inherently hierarchical, an object database like ObjectStore is the perfect data management system for XML. ObjectStore can store a tree-like structure as is - without the mapping code that is required by traditional, relational databases to convert the hierarchically structured object into rows and columns as shown in Figure 4. Being able to store XML natively also means that there are no run-time joins to hamper performance during execution. With ObjectStore, navigating an XML tree is simply traversing a pointer in local memory. It gives applications high-speed access to XML and relieves the developer from having to manage a complex object model as well.

Figure 4 Storing XML in ObjectStore



ObjectStore also has sophisticated index and query capabilities. When data is aggregated in the middle tier and cached in ObjectStore as XML, applications can do high-speed queries across the local XML datastore. The performance is high because the operation does not have to hit the back-end data source, and the results are reliable because it does not rely on out-of-date data from a batch process. ObjectStore delivers high-speed access as well as manipulation of XML data.

4.2 Extensibility

An XML datastore does more than boost performance. It supports the use of *dynamic data*, a job that cannot be done otherwise. As application servers share data objects, it is inevitable that they will need to update the data. These modifications need to be saved, and the data object has to change on the fly.

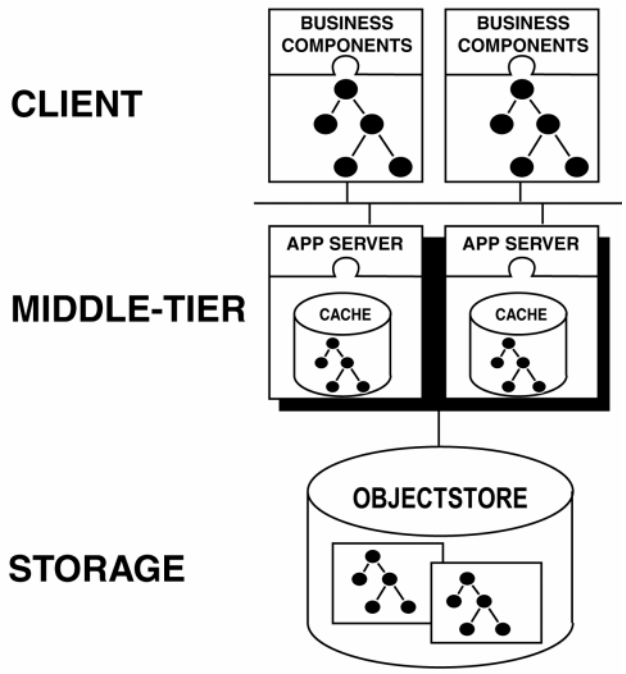
ObjectStore, used as an XML datastore, supports the manipulation and extensibility of XML data. Because ObjectStore can support a data abstraction model for XML, developers can add a new field or attribute to a data structure without having to redefine the object model or evolve the schema. ObjectStore updates the data object without disrupting the existing structure or breaking the application code. This is functionality that is not possible with relational databases, because the addition of a nested field requires a new object model superimposed on top of the added table columns.

4.3 Middle Tier Support

Another important requirement of an XML management system is the ability to distribute across the middle tier. If it is to act as a cache for application servers, the data at each server must be up to date and available with complete transactional consistency. ObjectStore, with its unique *Cache-Forward* architecture, does exactly that. It moves its cache forward – to the application servers – and uses the patented call-back locking algorithm to manage the data at each server. This is shown in Figure 5.

Figure 5

ObjectStore's Cache-Forward Architecture



ObjectStore automatically manages the data so that there is a virtual XML datastore in each application server, which frees developers from having to manage data caches by hand. Because the locks are also managed in the middle tier, round-trips to the database server are eliminated so there is no bottleneck even when many application servers are added to the middle tier. This supports extreme scalability, because additional application servers can be applied without rearchitecting.

For more information on *the Cache-Forward* architecture, refer to "Component-Based Computing and the ObjectStore Cache-Forward Architecture" available at http://www.objectdesign.com/papers.html.

To support connectivity with back-end data repositories, Object Design offers DBconnect. DBconnect provides user-controlled synchronization with legacy relational databases eliminating the need to write mapping code to translate rows and columns into XML. Alternatively, relational database vendors are developing their own XML adapters to do the translation. Once data reaches the middle tier as XML, it can be aggregated and managed by ObjectStore.

An XML data management system in the middle tier can support XML for the application server. It gives high performance access to the data while also simplifying application development. By having XML as the standard format for data interchange, application developers are relieved from writing and maintaining data integration code. ObjectStore, with its native support for XML and *Cache-Forward* architecture, offers a fast and secure datastore for server-side XML and provides a development platform for integration applications.

5.0 Conclusion

XML can do many exciting things. One of the most compelling usages is as a common denominator for data aggregation across application servers. XML, which is quickly becoming the standard for distributed data, makes the task of developing integrated systems easier by reducing the amount of custom data translation and interpretation code that has to be written. Well-managed server-side XML offers a simple and elegant solution that tames the middle tier Tower of Babel and allows developers to focus on their business logic.

ObjectStore, with its unique *Cache-Forward* Architecture, efficiently manages server-side XML and the intricacies of aggregating data from disparate sources because of its ability to manage structured data and distribute across the middle tier. By caching data as XML in the middle tier, application developers can look at all data through a unified view and execute high-speed queries and data access operations. XML is a common denominator for integrating data, and ObjectStore brings that data to the client with maximum speed and integrity.

Because of these unparalleled features, ObjectStore is the best technology for bringing the compelling benefits of XML to middle tier development. One of the great promises of XML is that it can simplify application development by solving the data integration problem - but to realize these benefits, XML must be supported with true data management. ObjectStore, with its ability to manage, serve, index and query native XML data, is the ideal foundation for any data integration solution. Object Design is committed to delivering such a solution by building on the strengths of ObjectStore and the versatility of XML.