Distributed Xbean Applications

Bruce E. Martin
jGuru
martin@jguru.com

Abstract

XML has emerged as the universal standard for exchanging and externalizing data. Software products of all kinds are being upgraded to "support XML." Typically this means they can import and export XML data.

But just defining standard representations for exchanging data is insufficient. The data need to be integrated with existing applications and databases and processed by programs written in some programming language.

This paper describes distributed applications constructed from Xbeans. Xbeans are Java Beans that manipulate XML data. With the appropriate set of Xbeans and a Java Bean design tool, it is possible to build useful distributed applications with little or no programming.

1 Introduction

The Extensible Markup Language[1], or XML, has emerged as the universal standard for exchanging and externalizing data. Software products of all kinds are being upgraded to "support XML." Typically this means they can import and export XML data.

At the same time, standards groups representing almost every human endeavor are agreeing upon XML Document Type Definitions (DTDs) for exchanging data. One of many examples is the International Press Telecommunications Council[2]; it has defined an XML DTD allowing "news information to be transferred with markup and be easily transformed into an electronically publishable format." These vertical market standards will allow diverse applications to exchange data in unforeseen ways.

But just defining standard representations for exchanging data is insufficient. The data need to be integrated with existing applications and databases and processed by programs written in some programming language.

To the end of accessing XML data from different programming languages, the W3C has defined the Document Object Model (DOM) standard[3]. The DOM is an application programmer's interface to XML data. It is available from many programming languages, including C, C++, Delphi, Java, Perl, Python and Tcl. As such, Java programs can access XML data via the DOM API.

Rather than structuring software that manipulates XML data as mammoth programs, *software component* technology allows developers to package smaller grained pieces of reusable functionality. *Java Beans[4]* are software components that support the packaging, reuse, connection and customization of Java code. Design tools allow applications to be created by connecting and customizing existing Java Beans.

2 Xbeans

Xbeans are Java Beans that manipulate XML data. With the appropriate set of Xbeans and a Java Bean design tool, it is possible to build useful distributed applications with little or no programming. (We will describe several applications later.)

As illustrated in Figure 1, an Xbean consumes XML as input, processes it in some fashion and then produces XML as output.

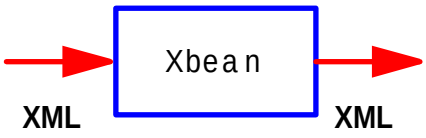

Figure 1. An Xbean consumes XML as input, processes it and then produces XML as output.

Xbeans consume and produce XML as DOM documents. That is, the data passed to Xbeans are not strings that need to be parsed by an XML parser, but an already parsed document object that is accessed via the w3c standard DOM API. As such, figure 1 is not precise. Figure 2 shows the Xbean processing more precisely.

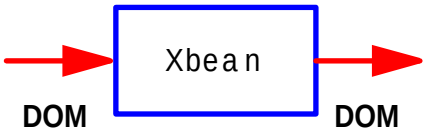

Figure 2: Xbeans consume and produce XML as DOM Documents.

As shown in Figure 3, Xbeans are connected to each other forming a *channel*. Data flow from one bean to another in a channel. This is very similar to a UNIX pipe[5]; typed XML data flow, rather than untyped bytes. Xbeans are also similar to CORBA event channels[6].

For now assume that only a single Xbean consumes the output of the previous; later we will describe an Xbean that acts as a parallelizer.

3 Applications of Xbeans

The Xbean paradigm is a very general data flow mechanism. XML describes structured data; Java provides the computation and control on the data. As described in detail later, Xbeans are functionally composable simply by supporting a couple of minimal interfaces.

We now illustrate the power of the Xbean paradigm with a few distributed application examples:

3.1 Data exchange between enterprises

Enterprises want to exchange data. Industry specific standards efforts are defining XML Data Type Definitions (DTDs). These DTDs represent the semantics and format of the data to be exchanged.

Enterprises, however, have their data in their own databases defined by existing schema. That is, no two enterprises represent the same data in the same way. The idea is to access native data, translate it according to a standard DTD, transport it, translate it according to a native DTD and finally store it.

Figure 4 illustrates a simple data exchange between enterprises using Xbeans. The boxes represent different Xbeans.

Each Xbean is configured appropriately. The *accessor* Xbean is configured to perform a particular SQL query and represent the result as an XML document. The *translator* Xbean is configured to translate the incoming XML document into an XML document that conforms to the agreed upon DTD for exchanging data. The *sender* and *receiver* Xbeans are configured to cooperate to transport the data.

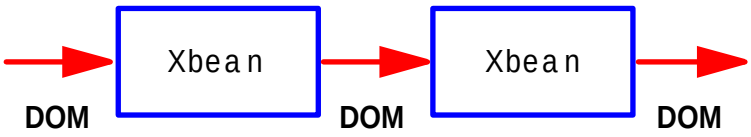

Figure 3. Xbeans are connected together into channels.

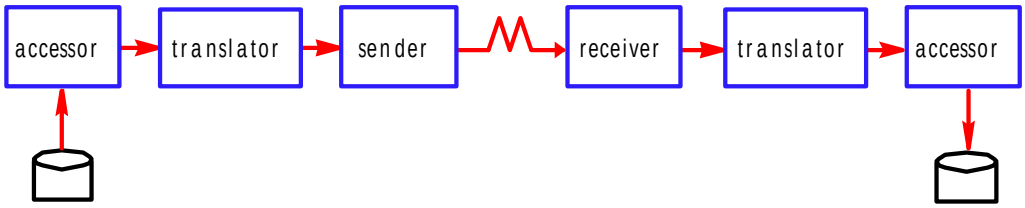

Figure 4: Data exchange between enterprises using Xbeans.

At the enterprise that receives the data, a configured *translator* translates the data from the agreed upon DTD to a DTD that more closely matches the native schema. Finally the *accessor* is configured with an SQL query that stores the incoming data appropriately.

3.2 Purchase order

An application of the data exchange given above is a purchase order. In this case the accessor Xbean is configured to perform an SQL query to obtain the data from various tables. The translator is configured to translate the data to conform to the standard DTD for a purchase order and so on.

3.3 Distributed Work flow

With a rich set of configured Xbeans that represent control flow and interact with users, Xbeans can be configured to create distributed work flow applications.

3.4 Web News Channels

Many web sites are now publishing news and other information as XML of various DTDs so that other web sites can "rebroadcast" it. Such distributed service could be implemented using the appropriate set of Xbeans.

4 Xbean Interfaces

To be an Xbean, at least one of two simple Java interfaces must be supported. Any Java bean that implements at least one of these interfaces is an Xbean. The interfaces extend the Java event interfaces. The interfaces are slight variations of the interfaces defined by IBM in their AlphaWorks XML Productivity Kit for Java[7]. The interfaces were changed to better support distributed applications.

The *DOMListener* interface, as given in figure 5, defines a single operation, *documentReady(DOMEvent evt)*, for passing the XML document to the Xbean.

```
public interface DOMListener
            extends EventListener {

    public void documentReady(
                DOMEvent evt)
           throws XbeansException;
}
```

Figure 5. The DOMListener interface.

The *DOMSource* interface, as given in figure 6, defines two operations, *getDOMListener() and setDOMListener(DOMListener next)*, for getting and setting the next Xbean, i.e. the Xbean that will receive the output of the Xbean. The *DOMSource* interface does not directly support the addition of multiple listeners, most Xbeans only support the registration of a single listener to avoid hidden concurrency issues. The parallelizer Xbean supports multiple listeners, making the concurrency issues apparent at configuration time.

```
public interface DOMSource {
   public void setDOMListener(
        DOMListener DOMListener);
   public DOMListener getDOMListener();
}
```

Figure 6. The DOMSource interface.

4.1 Source and Sink Xbeans

As illustrated in figure 7, an Xbean that *implements* the *DOMSource* interface and *uses* the *DOMListener* interface is called a "source Xbean", that is it is a source of XML data.

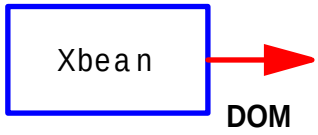

Figure 7. A "source Xbean" implements the DOMSource interface and uses the DOMListener interface.

An Xbean that implements the DOMListener interface is called a "sink Xbean", that is it receives XML data.

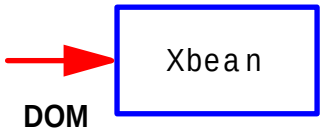

Figure 8. Any Java Bean that implements the DOMListener interface is a "sink Xbean".

For maximum flexibility, most Xbeans are both sources and sinks of XML data. (See figure 2.)

4.2 Generic Xbeans

Generic Xbeans process any kind of XML document using the DOM API. Many generic Xbeans are configured using the standard Java Bean mechanisms of property editors and customizers.

There are many useful generic Xbeans possible. Below are some generic Xbeans that are useful in many applications.

accessor

The *accessor* Xbean is configured to perform a particular SQL query and represent the result as an XML document. The accessor can also be configured to receive an XML document and store it according to an SQL query.

translator

The *translator* Xbean translates an XML document from one arbitrary DTD to another. It includes a graphical Java Bean customizer for easily specifying the translation.

sender
receiver

The *sender* and *receiver* Xbeans*[13]* are configured to transport XML data over a network. Different implementations of the sender-receiver are possible, offering different qualities of service and using different transport mechanisms.

viewer

The *viewer* Xbean graphically displays the content of an XML document.

parser

The *parser* Xbean converts an XML document in its textual format into a DOM document. The parser is customized with the name of the source of the document.

writer

The *writer* Xbean converts a DOM document and externalizes it into its XML textual format. The writer is customized with the name of the target XML.

logger

The *logger* Xbean logs every time an XML document passes through it.

parallelizer

The *parallelizer* Xbean receives a single DOM document and passes it to multiple listeners. For each listener, the parallelizer can be customized to specify whether the document is copied or shared among the listeners.

synchronizer The *synchronizer* Xbean receives multiple DOM documents and passes on another DOM document to its listener.

4.3 Document type specific Xbeans

An Xbean can also be specific to a particular XML document type, that is it can be programmed to only work on XML documents whose type is known at compile time. While less general, it can still be part of a channel. Type specific Xbeans must check the type of the incoming XML document to ensure type integrity. Generic Xbeans can receive and process the output of a type specific Xbean.

A type specific Xbean can be written using the DOM interface. However, the DOM is usually overkill for manipulating XML documents whose type is known at compile time. Sun, through their Java Community Process, is trying to simplify access to XML data for such documents by defining a standard Java XML Data Binding[14]. The idea is to specify a compiler that would map a DTD or an XML schema to a Java class that represents the data. Unlike the DOM, which provides a more complex, generic API to any XML document, the compiler would effectively define a schema specific API at compile time. Presumably this will make Java code that needs to manipulate an XML document of a particular type much easier to write.

5 Configuring Xbeans into Distributed Applications

Source Xbeans have a *DOMListener* property, as specified in the *DOMSource* interface. Thus connecting Xbeans simply means setting the property to be the next Xbean. Source Xbeans invoke *documentReady* on the next Xbean.

Xbeans can be configured using the standard Java Bean mechanisms of property editors and customizers. Using a Java Bean design tool, such as IBM's Visual Age for Java[8], Inprise's JBuilder[9], Symantec's Visual Cafe[10], FreeBuilder[11] or NetBeans/Forte[12], a developer can visually instantiate, customize and connect Xbeans. Complete distributed applications can be created, often without writing any code.

Besides Java IDE design tools, more simplified tools that only allow the configuration and customization of Xbeans are desirable since they allow non-Java experts to configure and customize Xbeans into useful application channels.

6 Sending XML "over-the-wire"

As illustrated in figure 9, the sender and the receiver Xbeans completely abstract the communication of XML "over the wire."

Figure 9. The sender and receiver Xbean abstract the communication of XML over the wire.

There are many possible implementations of the sender and receiver. The sender and receiver must agree on a transport mechanism, an external representation of the DOM over the network and other qualities of service. We investigated three different approaches to implementing the sender and the receiver. After presenting the three approaches, we describe the performance characteristics of each approach.

Figure 10 illustrates an implementation that uses HTTP as a transport mechanism and represents the DOM as external XML text.

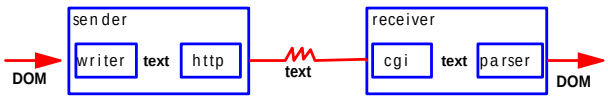

Figure 10. A sender and receiver Xbean that use HTTP as a transport and represent the DOM as XML text over the network.

Figure 11 illustrates implementations of the sender and receiver that use Java RMI as a transport and represent the DOM using Java serialization. The DOM implementation must support Java serialization for this implementation to work.

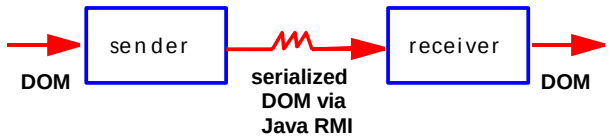

Figure 11. A sender and receiver Xbean that use Java RMI as a transport and Java serialization of the DOM to represent the DOM over the network.

The final implementations of the sender and receiver are illustrated in figure 12. The sender and receiver use CORBA as a transport mechanism and the XML is represented natively by CORBA. Such implementations do not yet exist. Future versions of CORBA will support XML as a native data type. For now, the sender and receiver use CORBA as a transport and Java serialization to represent the DOM over the network.

Figure 12. Future versions of CORBA will support XML as a native network data type. For now, the implementation of the sender and the receiver use CORBA as a transport mechanism and Java serialization to represent the DOM over the network.

6.1 Preliminary Performance Results

We did space and execution time performance tests to compare the three different implementations of the sender and receiver Xbeans..

The space required to represent an XML document depends on the structure implied by the DTD and the amount of data in the document. Obviously, an XML document with one simple tag and megabytes of textual data is going to have almost identical space requirements in both representations.

We are interested in highly structured data. Our performance tests compared the XML textual representation of a purchase order to the serialized DOM representation of it. Our performance tests use IBM's XML parser / DOM implementation, version 2.0.13.

We used a purchase order since it is a typical business-to-business application of XML and has a fairly rich XML structure. It represents information extracted from several tables in a relational database. Figure 13 gives an XML document containing a single purchase order.

```
<orders>
  <order>
    <header>
      <billing_info>
        <name>
          <given>John</given>
          <family>Doe</family>
        </name>
        <address>
          <street>555                Main
Street</street>
          <city>Mill Valley</city>
          <state>California</state>
          <zipcode>94520</zipcode>
          <country>USA</country>
          <phone>707 555-1000</phone>
        </address>
        <credit_card>4555
5</credit_card>
      </billing_info>
     <shipping_info>
        <name>
          <given>John</given>
          <family>Doe</family>
        </name>
        <address>
          <street>555                Main
Street</street>
          <city>Mill Valley</city>
          <state>California</state>
          <zipcode>94520</zipcode>
          <country>USA</country>
          <phone>707 555-1000</phone>
        </address>
```

```
      </shipping_info>
    </header>
    <item>
      <product_id>5555555</product_id>
      <prod_name>Widget</prod_name>
      <quantity>100</quantity>
      <price>.25</price>
    </item>
    <total>25.00</total>
  </order>
</orders>
```

Figure 13. A single purchase order in XML.

To see how each sender-receiver implementation behaves as the amount of XML data gets larger, we simply replicated the number of purchase orders in the XML document. Figure 13 gives a single purchase order. A document with multple purchase orders includes the `<order>` … `</order>` data multiple times.

The first test we did was to compare the space required for representing the XML document as text vs. the space required for representing the XML document as a serialized DOM. Table 1 gives the space results.

Table 1. Space test results

Number of Orders	Bytes to represent as XML text	Bytes to represent as a serialized DOM	Ratio (serialized / text)
1	1,048	7,278	6.9
5	4,900	29,310	5.9
10	9,715	56,850	5.9
100	96,385	552,570	5.7
500	481,585	2,755,770	5.7

Surprisingly, the XML text representation was much more efficient than the serialized DOM.

The next test we did was to compare the execution time required for converting between the DOM and the respective on-the-wire formats. In particular, we compare the time it takes to externalize a DOM representation to XML text and then reparse it into a DOM vs. the time it takes to serialize and deserialize the DOM representation. This effectively compares the performance of the senders and receivers, without considering the actual communication of the data.

The performance tests were performed on a Compaq Pentium 3 system running at 450 Mhz with 128 MBytes RAM. The system runs Microsoft Windows NT 4.0, Sun's Java 1.2 Virtual Machine and IBM's XML parser / DOM implementation, version 2.0.13. The timings are the average of 50 trials. Table 2 gives the execution results.

Table 2. Timing test results

Number of Orders	Milliseconds to write and reparse DOM	Milliseconds to serialize and deserialize DOM	Ratio (serialize / parse)
1	191	478	2.5
5	199	534	2.7
10	231	603	2.6
100	862	2228	2.6
500	4707	10107	2.1

Surprisingly, externalizing the DOM as XML text and then reparsing it to produce a DOM was much more efficient than the serializing and deserializing the DOM. Until there are better implementations of Java serialization and of the DOM, our tests indicate that it is better in both time and space to send XML text over the wire.

7 Xbeans.org

Xbeans.org is an open source project. The goal of Xbeans.org is to provide a rich repository of freely available Xbeans. The first version of Xbeans is available from the web site. Since Xbeans are clearly partitioned pieces of functionality with two well defined interfaces, independent and parallel development of Xbeans is greatly simplified. Unlike some open source projects, there is not a lot of coordination and project management required.

Xbean.org is looking for contributors of the Xbeans outlined here as well as any useful Xbean imaginable.

8 Conclusions

We have presented the Xbean model, described serveral example Xbeans and described how distributed applications that process XML data can be constructed using Xbeans. Additionally, we have given some preliminary performance results comparing three different implementations of the sender and receiver Xbeans.

9 References

1. Extensible Markup Language (XML) 1.0 W3C Recommendation 10-February-1998. http://www.w3.org/TR/1998/REC-xml-19980210

2. NewsML - Markup for the third millennium, International Press Telecommunications Council web site: http://www.iptc.org

3. Document Object Model (DOM) Level 1 Specification Version 1.0 W3C Recommendation 1, October, 1998. http://www.w3.org/TR/REC-DOM-Level-1

4. JavaBeans Specifications for Java 2, Sun Microsystems. http://java.sun.com/beans/glasgow

5. The UNIX Operating System. AT&T Bell Labs.

6. Bruce Martin, Yeturu Aahlad, Mod Marathe and Chung Le. “Asynchronous Notifications Among Distributed Objects.” In Proceedings of the 2nd USENIX Conference on Object-Oriented Technologies and Systems, June 17, 1996, Toronto, Canada.

7. AlphaWorks XML Productivity Kit for Java. http://www.alphaworks.ibm.com

8. IBM's Visual Age for Java Software Product, International Business Machines. http://www.ibm.com

9. Inprise's Jbuilder 3.0 Software Product, Inprise Corporation. http://www.inprise.com/jbuilder

10. Symantec's Visual Café Software Product, http://www.symantec.com

11. FreeBuilder Java IDE Software, http://www.freebuilder.org/index_org.html

12. NetBeans Developer Software Product. Sun Microsystems. http://www.netbeans.com

13. Bruce Martin, “Build distributed applications with Java and XML” JavaWorld, February, 2000. http://www.javaworld.com/javaworld/jw-02-2000/jw-02-ssj-xml.html

14. JSR-000031 XML Data Binding Specification (review closed August 23, 1999). Sun Microsystems. http://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.html

15. The Xbeans.org web site. http://www.xbeans.org