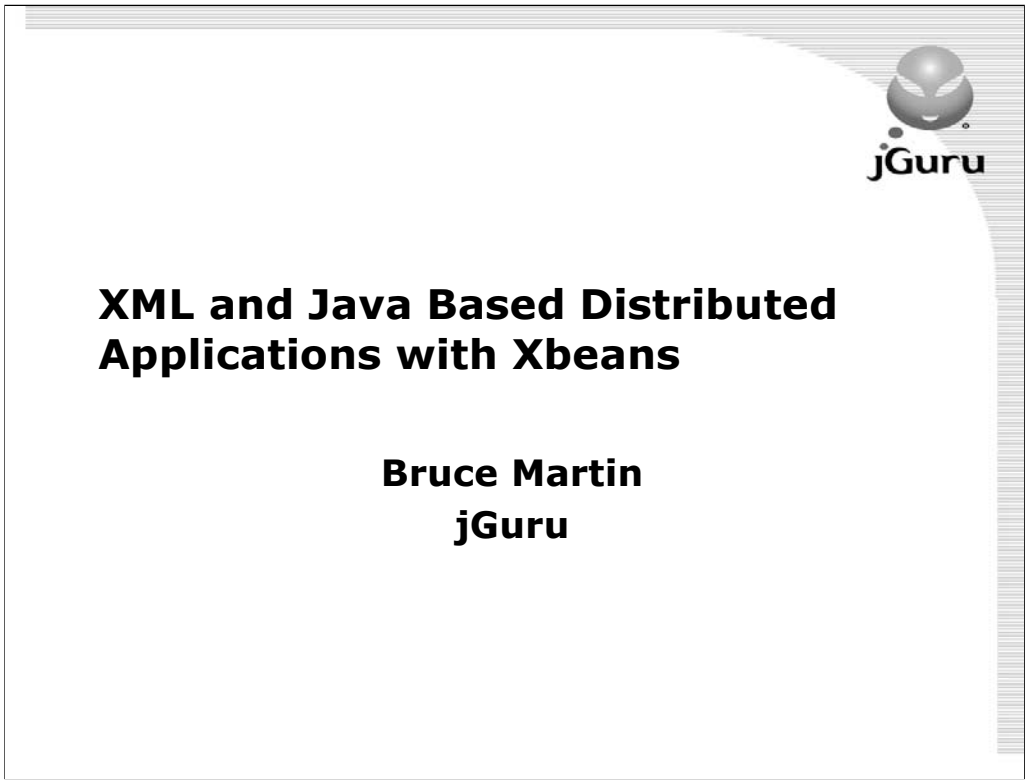
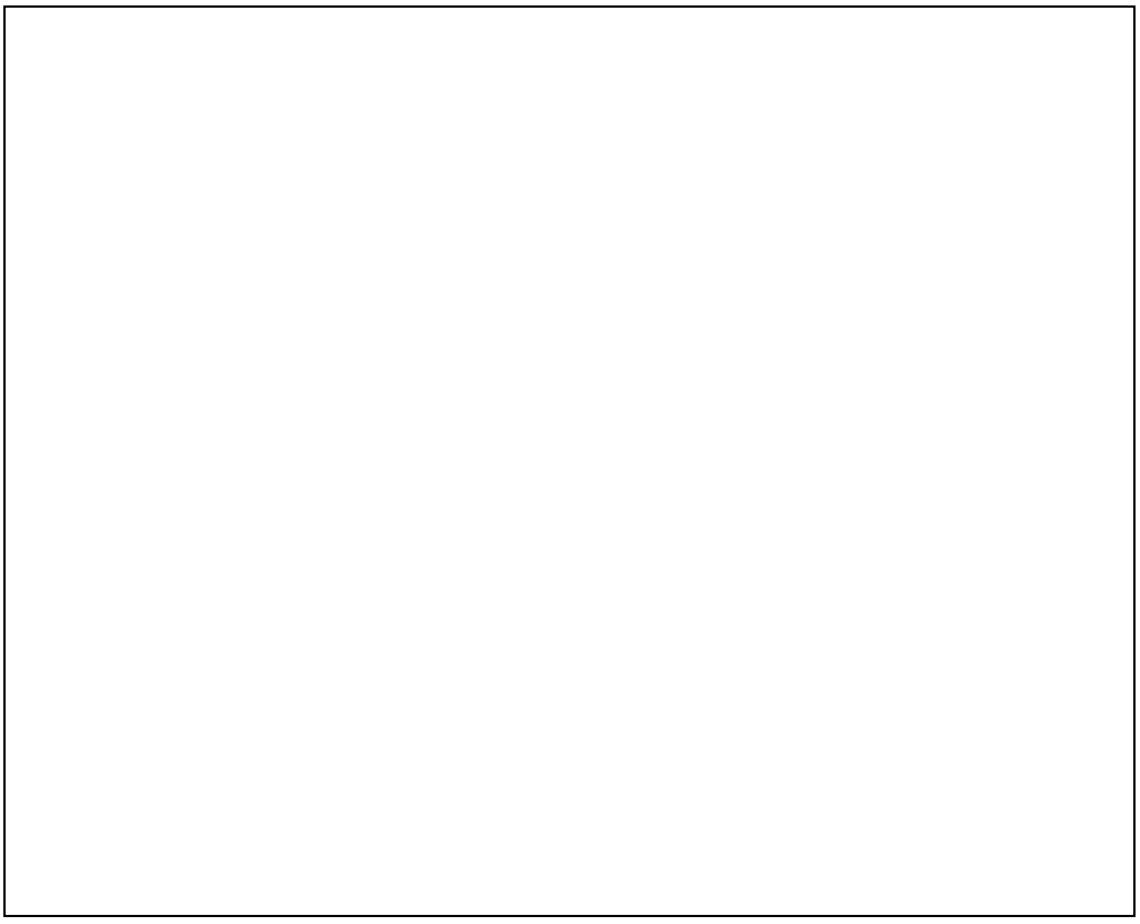

jGuru
XML and Java Based Distributed Applications with Xbeans
Bruce Martin
jGuru



Outline

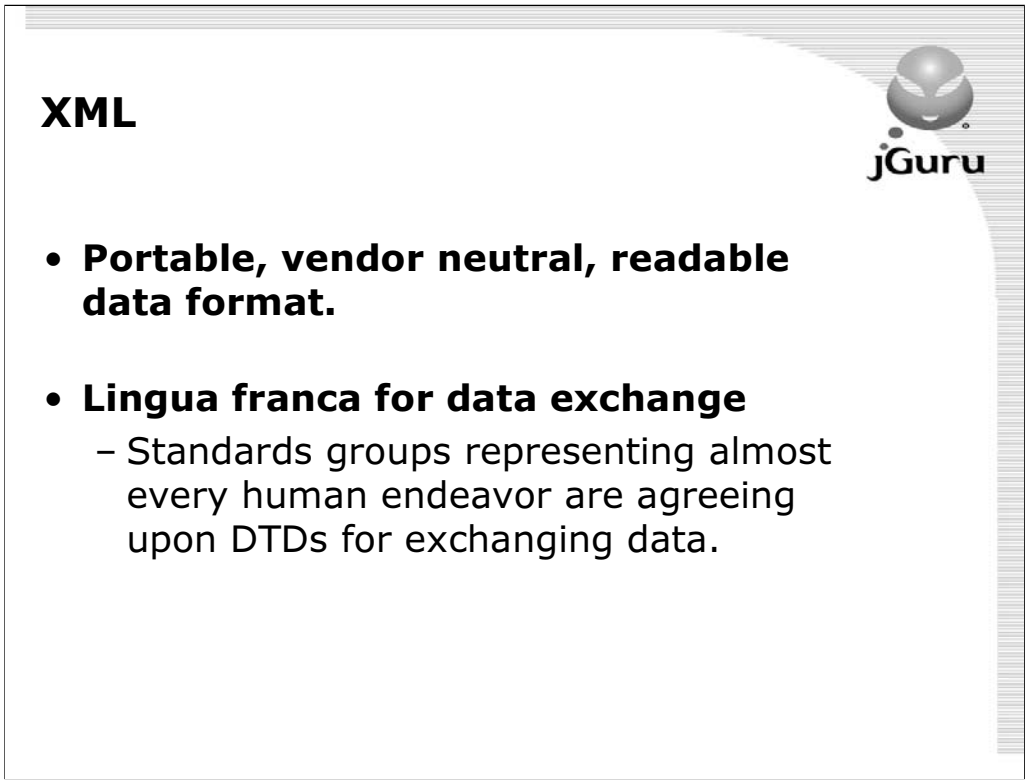
- **XML and distributed applications**
- **Xbeans defined**
- **Xbean channels**
- **Xbeans as Java Beans**
- **Example Xbeans**
- **XML over the wire**
- **Xbeans.org**

jGuru XML FAQ

- **www.jguru.com**
  - Currently 33 FAQs, including an XML FAQ
- **Questions and answers posed by jGuru community**
- **The FAQs have become quite interactive. jGuru >22K members.**

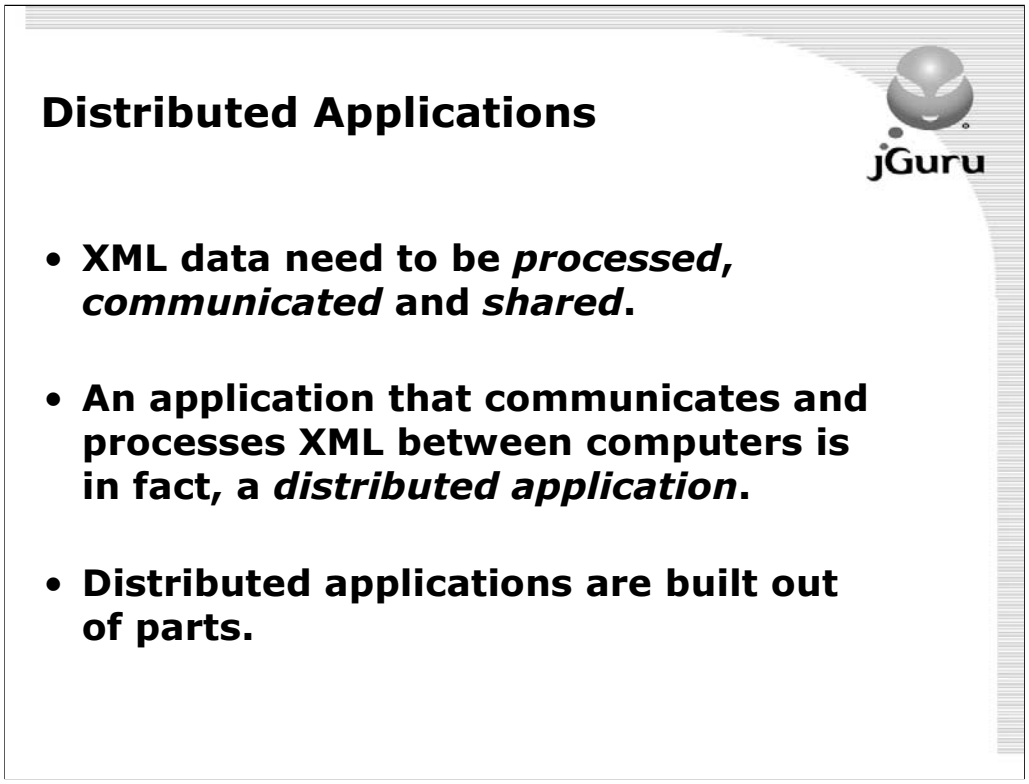
My favorite XML question:

Why would XML be a better solution than a typical delimited text file for B2B, or representing data in general? I can take a flat file, I can parse it and extract data, and then present that data in different ways using Java such as HTML or WML. So, why would I want to complicate things with DTDs, parsers, and APIs needed to extract and display data?



The Extensible Markup Language, 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; 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.

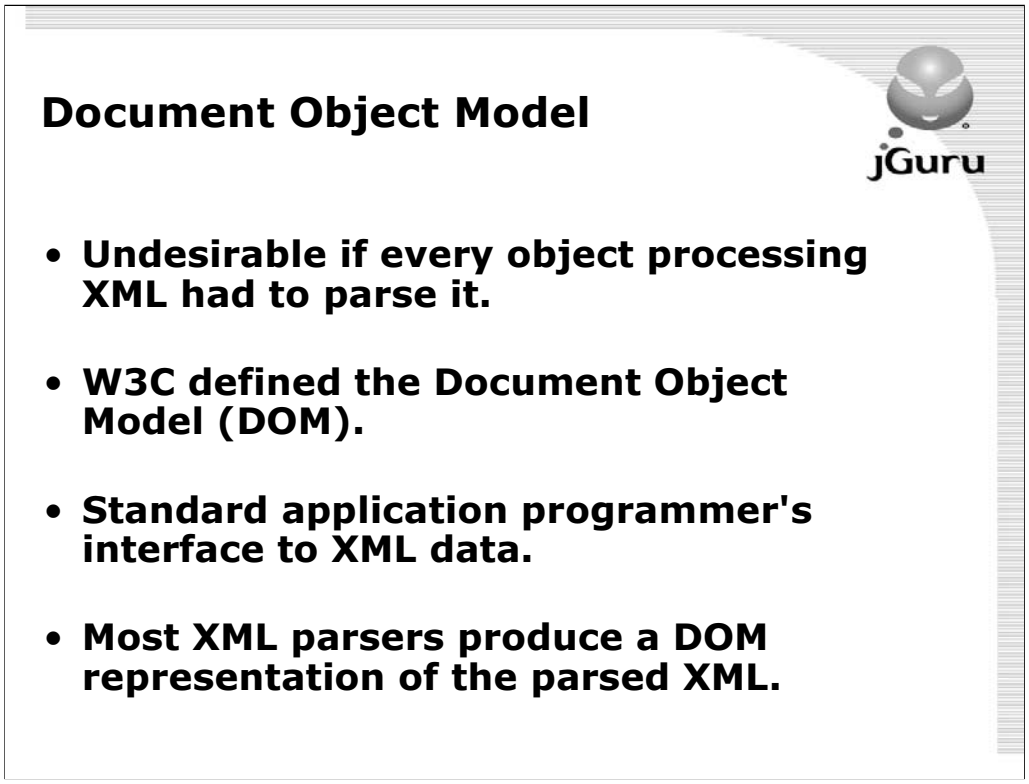
Rather than structuring software that manipulates XML data as mammoth programs, *software component* technology allows developers to package smaller grained pieces of reusable functionality. *Java Beans* are software components that support the packaging, reuse, connection and customization of Java code.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

Xbean

- **Java Bean that consumes XML, processes it and passes it on.**

- **Inspired by IBM's XML Productivity Kit for Java:**
  - Distributed application focus
  - Open source project

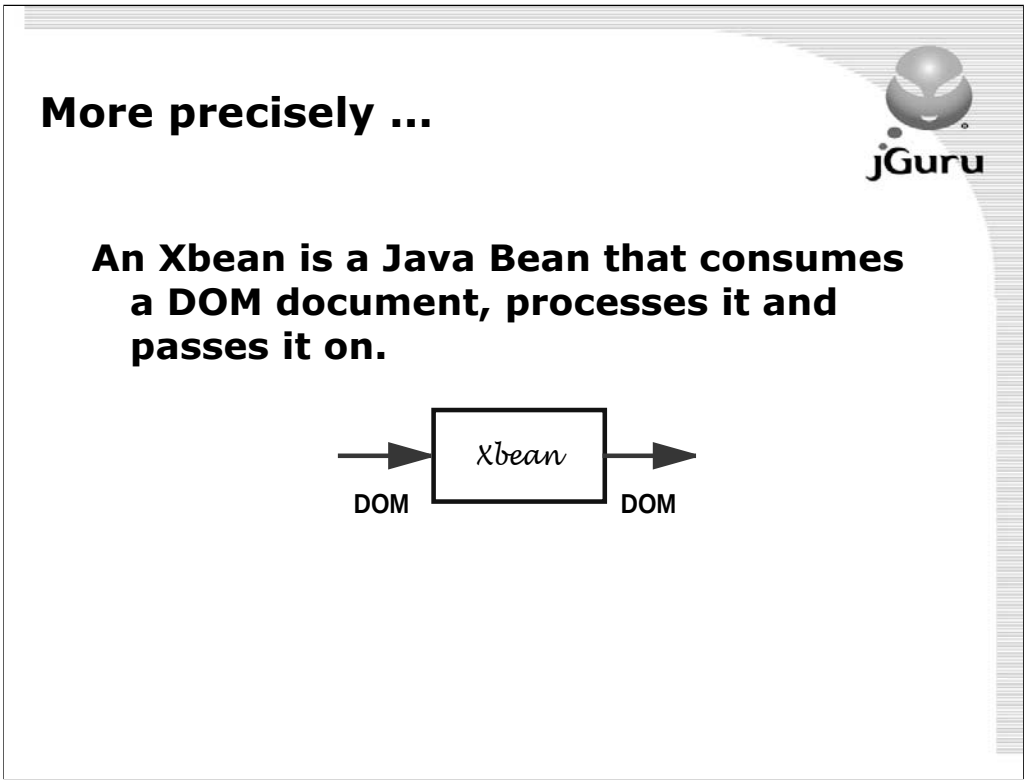
An Xbean consumes XML as input, processes it in some fashion and then produces XML as output.



To the end of accessing parsed XML data from different programming languages, the W3C has defined the Document Object Model (DOM) standard. The DOM is an application programmer's interface to XML data. It is available from many programming languages, including Java. Thus, Java programs can access XML data via the DOM API.

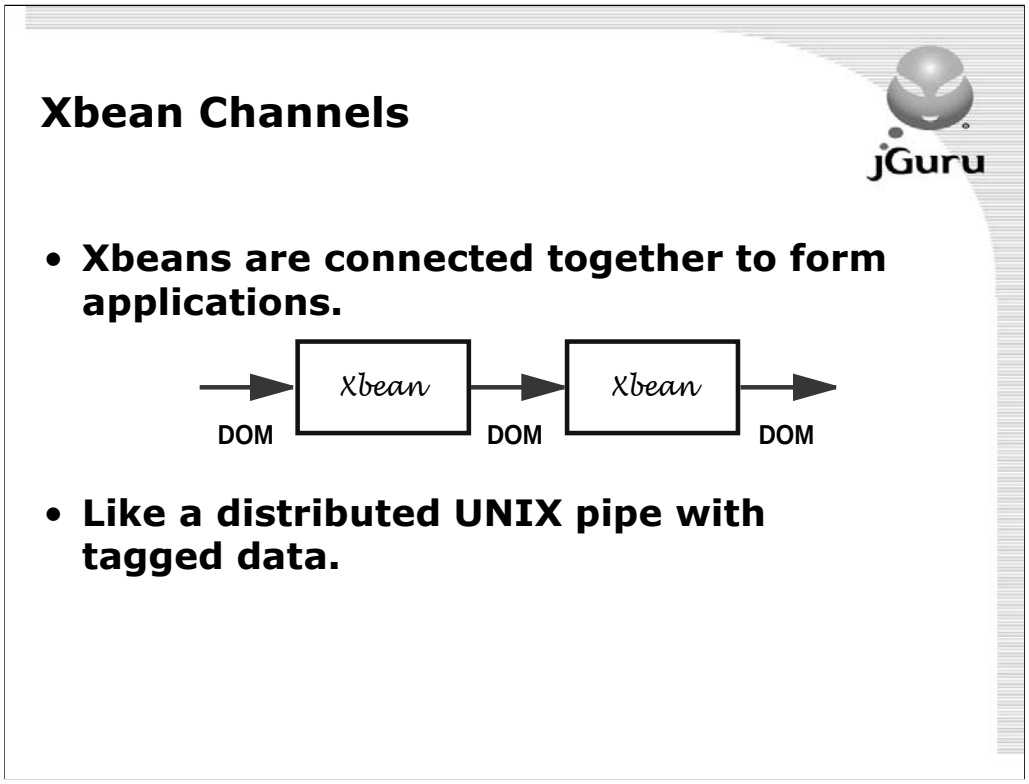
XML parsers produce a DOM representation of the XML document.

More precisely ...



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, the previous graphic of an Xbean is not precise. This graphic shows the Xbean processing more precisely.

Xbean Channels

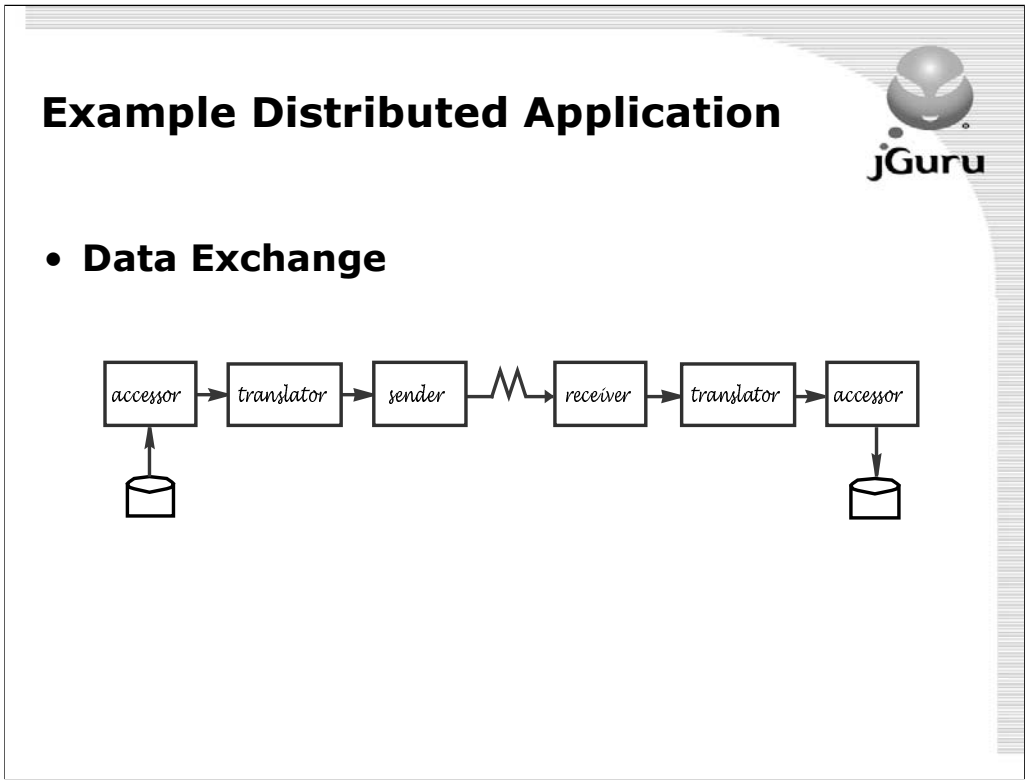


Xbeans are connected to each other forming a *channel*. Data flow from one bean to another in a channel. This is very similar to a UNIX pipe; typed XML data flow, rather than untyped bytes. Xbeans are also similar to CORBA event channels.

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.

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.



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.

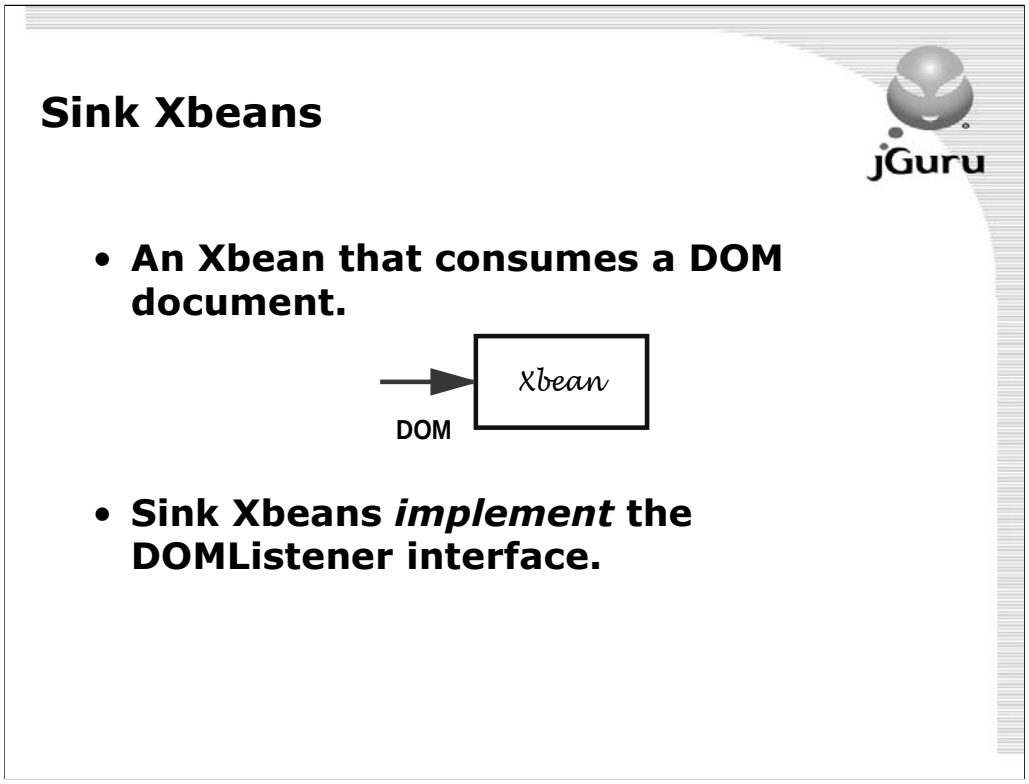
Enterprises, however, have their data in their own databases defined by existing schema. That is, no two enterprises represent the same data in the same way. The idea is to access native data, translate it according to a standard DTD, transport it, translate it according to a native DTD and finally store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

This slide illustrates a simple data exchange between enterprises using Xbeans. The blue boxes represents 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.

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.

Sink Xbeans



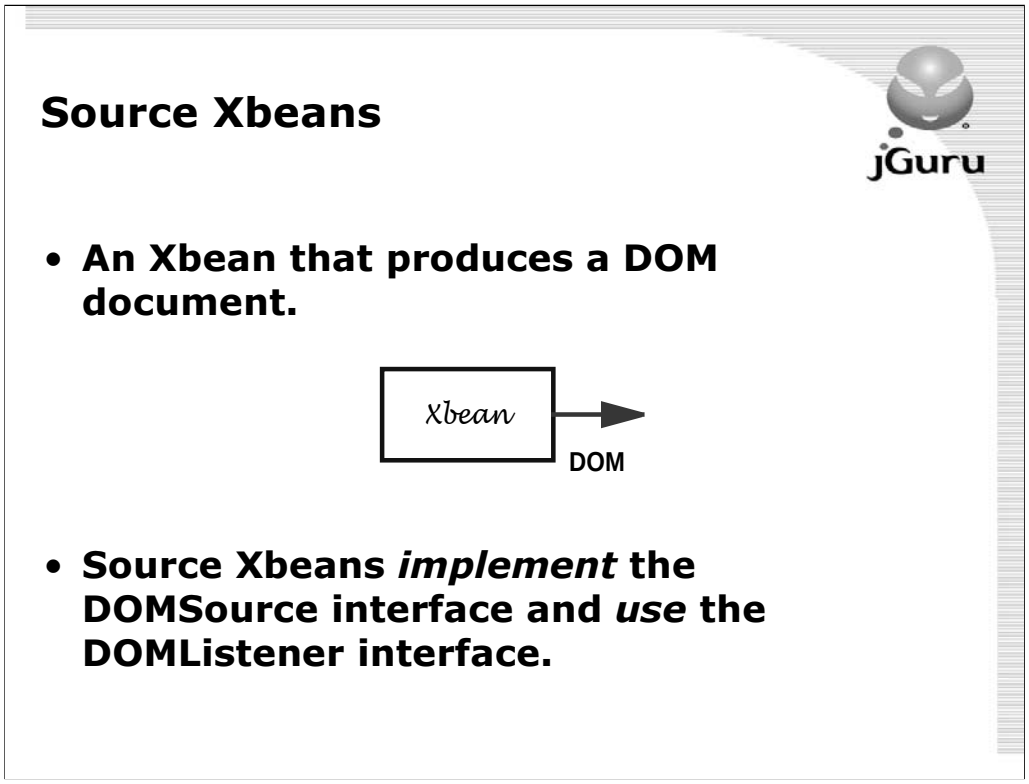
An Xbean that implements the DOMListener interface is called a "sink Xbean", that is it receives XML data via that interface.

Interface supported by a Sink Xbean


```
public interface DOMListener
   extends EventListener {

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

Here's the DOMListener interface. It has a single documentReady operation. The argument is the DOM document. This interface conforms to JavaBean standards for events.



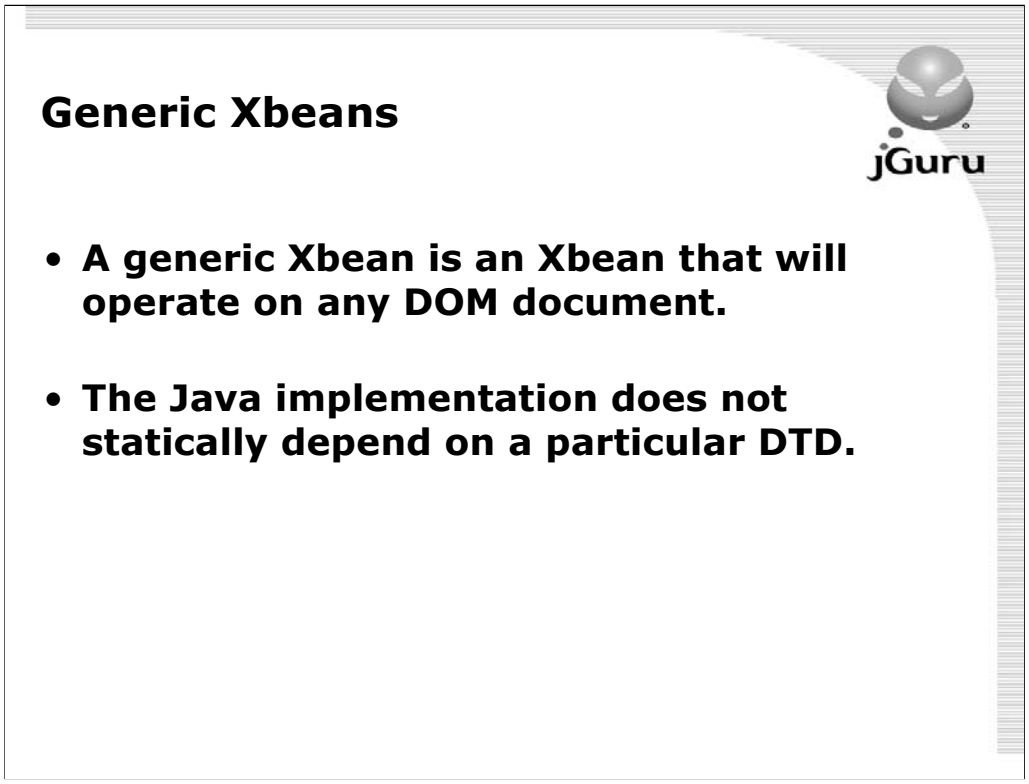
An Xbean that *implements* the *DOMSource* interface and *uses* the *DOMListener* interface is called a "source Xbean", that is it is a source of XML data.

Interface supported by a Source Xbean

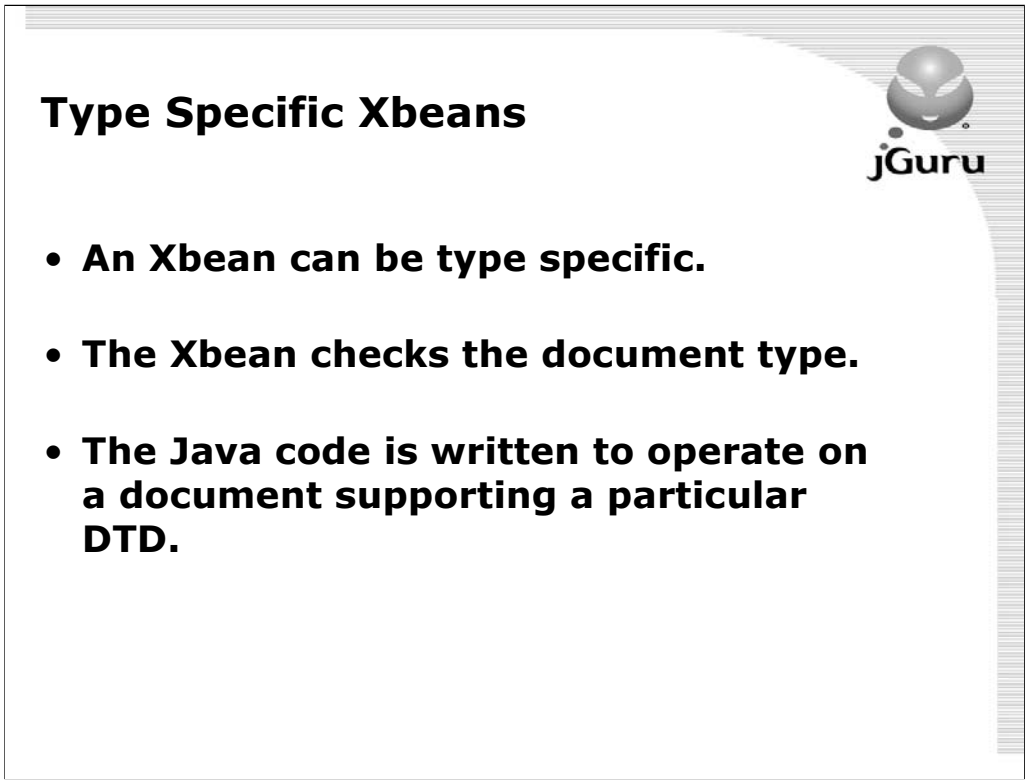

```
public interface DOMSource {
  public void setDOMListener(
                DOMListener DOMListener);
  public DOMListener getDOMListener();
}
```

- **Most Xbeans are both sources and sinks.**

Here's the DOMSource interface. It has operations for getting and setting the next Xbean in the chain. These operations correspond to the Java property DOMListener.



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



Xbeans 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, they 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.

Customized Xbeans

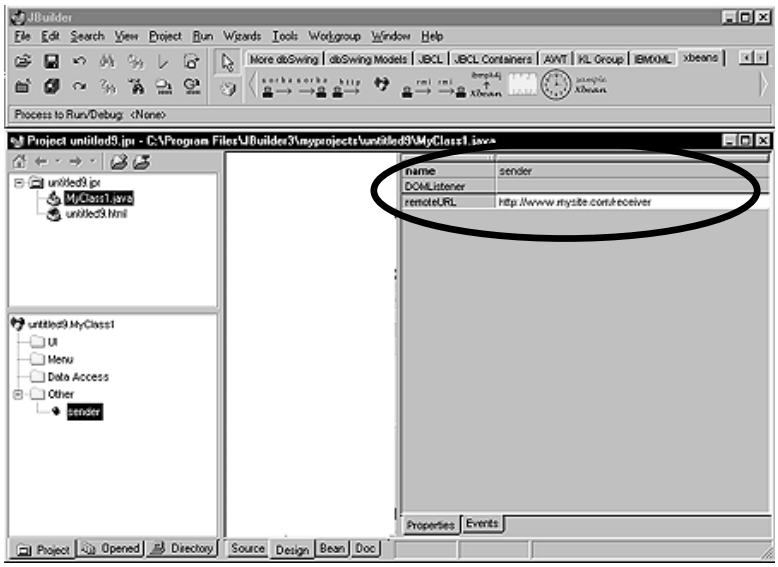
- **A customized Xbean is a generic Xbean that is customized at *design time* to operate on a specific type of XML document.**
- **The code is written generically but at runtime it operates on documents supporting particular DTDs.**

Xbeans are Java Beans

- **Xbean distributed applications can be created and configured in Java IDEs with Java Bean design tools:**
  - Visual Age
  - JBuilder
  - Café
  - NetBeans/Forte
  - FreeBuilder
  - …

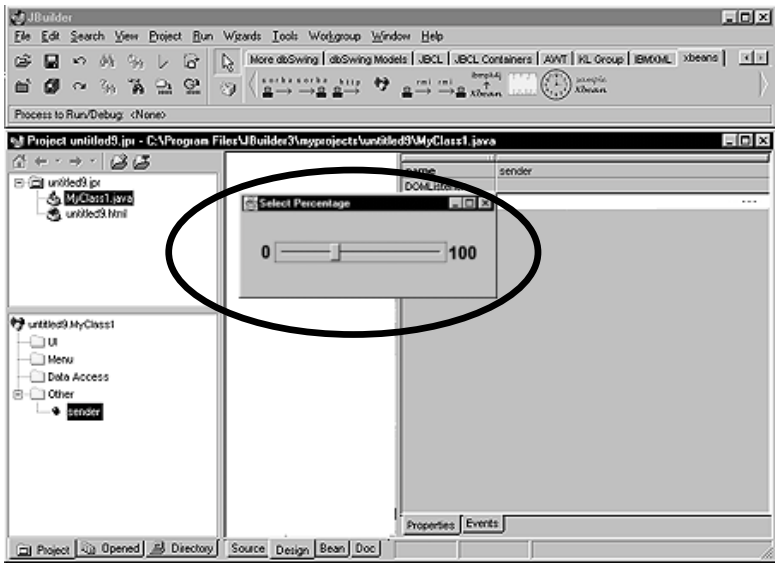
Xbeans can be configured using the standard Java Bean mechanisms of property editors and customizers. Using a Java Bean design tool, such as IBM's Visual Age for Java, Inprise's JBuilder, Symantec's Visual Cafe, FreeBuilder or NetBeans/Forte, a developer can visually instantiate, customize and connect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

Configuring Xbeans using a property editor



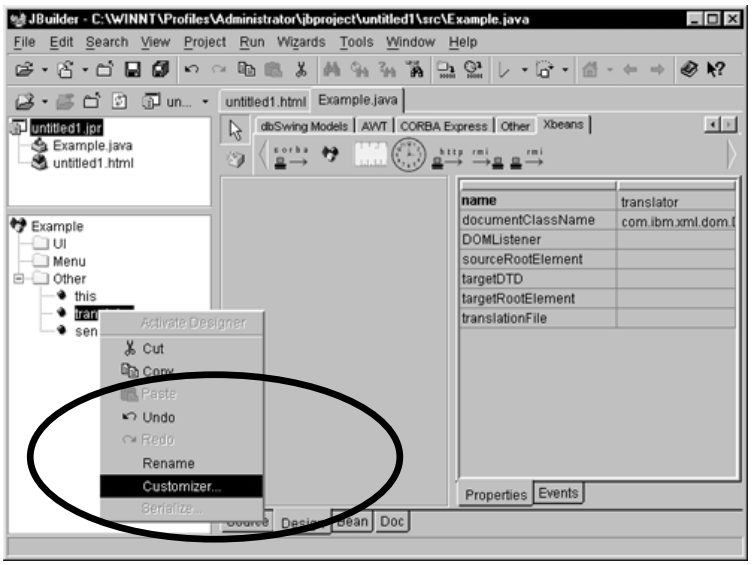
In the simplest case, the Java IDE's property editor can be used to set a property value at design time. This is an example of setting a property using Inprise's Jbuilder. Similar functionality exists in other Java IDEs.

An Xbean with its own property editor

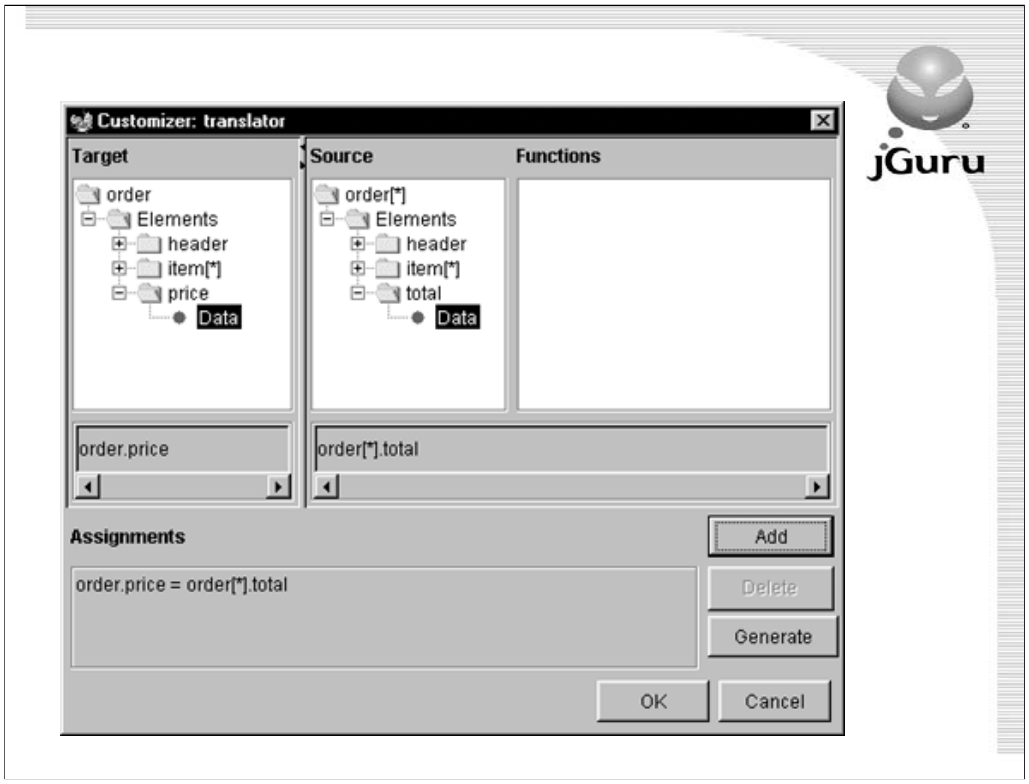


Some beans defined their own property editors. The editors can be invoked from the IDE to set a property value at design time. Again, this is an example of setting a property using Inprise's Jbuilder. Similar functionality exists in other Java IDEs.

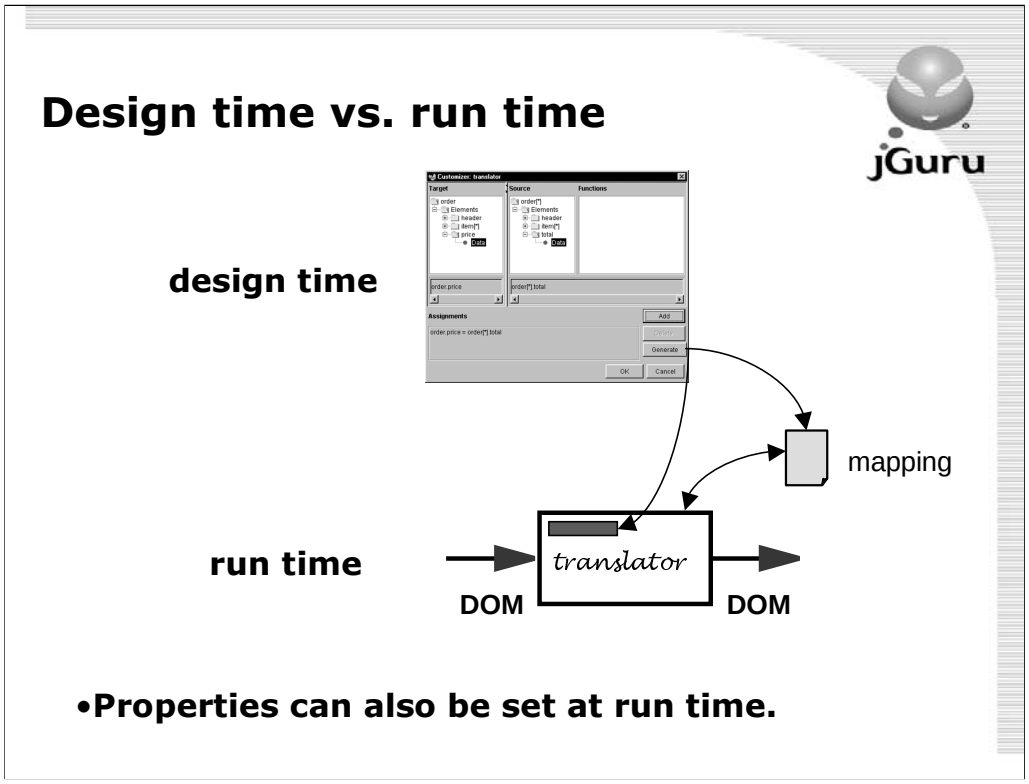
An Xbean with its own customizer



A customizer is used when more sophisticated bean configuration is needed. For example, if several properties are set from a complex user interaction. Customizers can be invoked directly from the IDE at design time. Again, this is an example of invoking a customizer using Inprise's Jbuilder. Similar functionality exists in other Java IDEs.

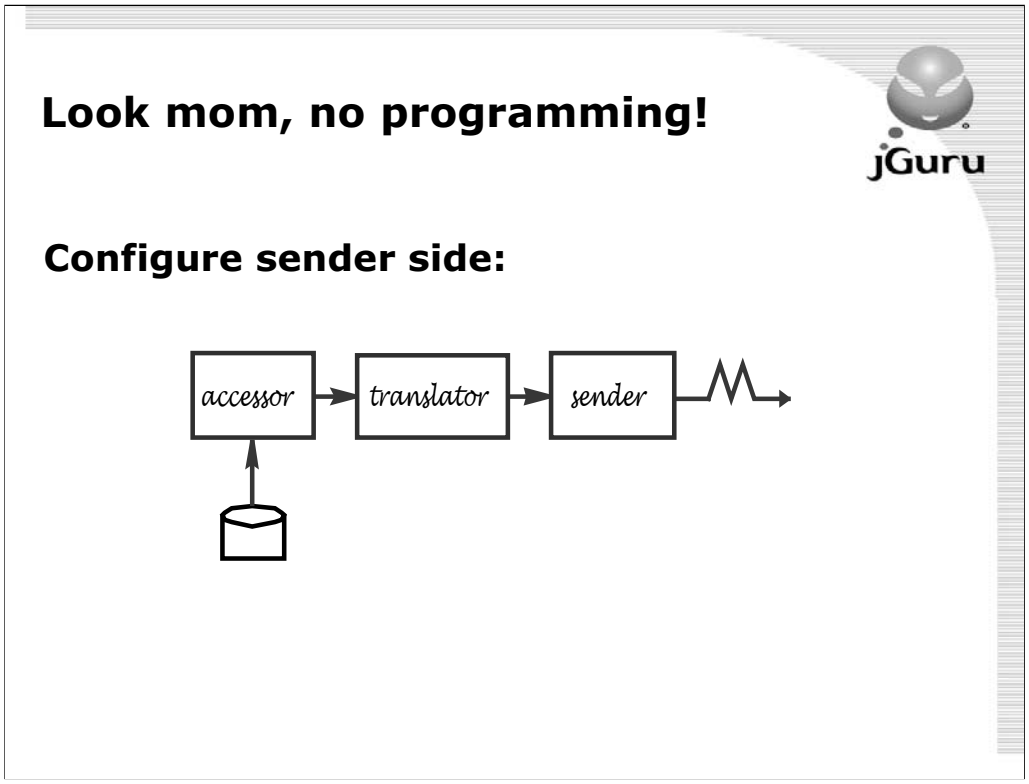


Here is the customizer example. In this case, the translator Xbean provides a GUI for specifying the mapping between a source document and a target document. At design time, the user configures the translator. Once the mapping has been specified, the user generates the mapping document and quits the customizer.



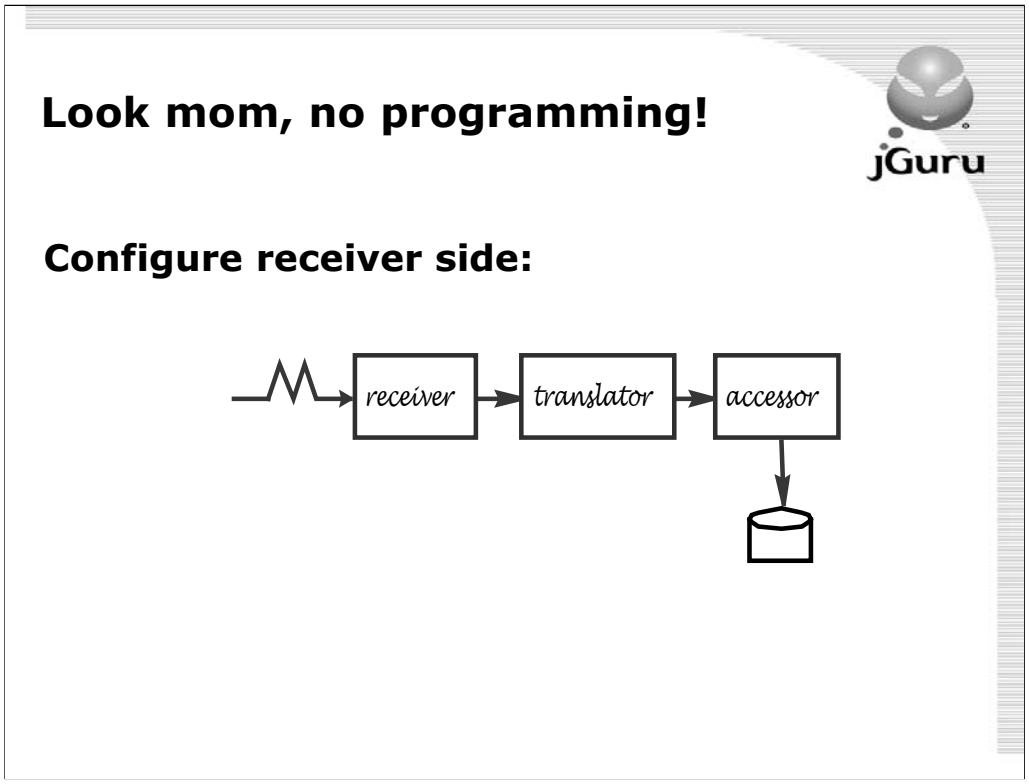
This slide summarizes the difference between configuring the Xbean at design time and the Xbean processing at run time. The customizer is invoked at design time to configure the Xbean. The result of this is a mapping document which describes the translation. The customizer sets a property in the Xbean that specifies the mapping document. At run time, the translator Xbean processes the mapping document to configure itself. When the translator receives a DOM document it translates it according to the mapping.

Note that this configuration is not just limited to design time. At run time, the mapping property could be set on the translator Xbean. This would cause it to process a different mapping document and make different translations.



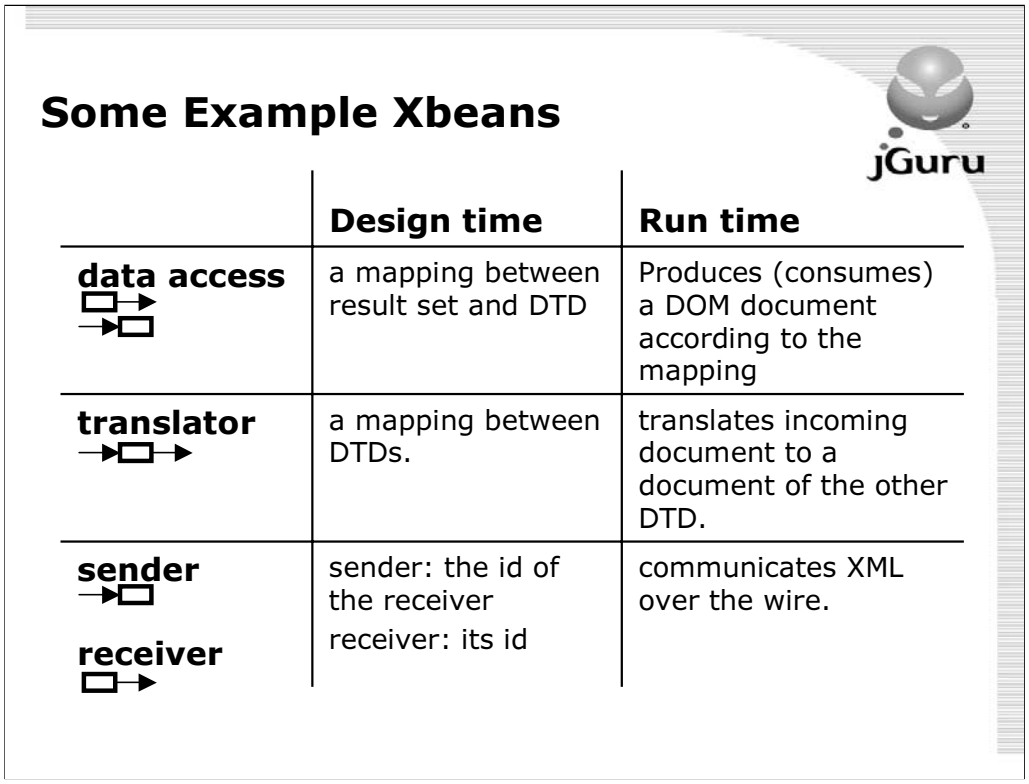
Given the right set of Xbeans and a Java Bean design tool, a simple distributed data exchange application can be built without any additional programming.l

On the sending enterprise side, an accessor Xbean is configured to query a data source, a translator Xbean is configured to translate the data to an agreed upon DTD and the sender Xbean is configured with the identity of the receiver.



On the receiving enterprise side, the receiver Xbean is configured with its identity so that it will advertise its services at run time. Similarly, a translator Xbean is configured to translate the data from an agreed upon DTD to a format acceptable to the accessor Xbean. The accessor Xbean is configured to update the data source according to the receiving enterprise's schema.

Of course, other application specific logic can be added by adding other Xbeans. This shows the basic communication architecture for the data exchange application.

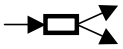
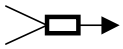


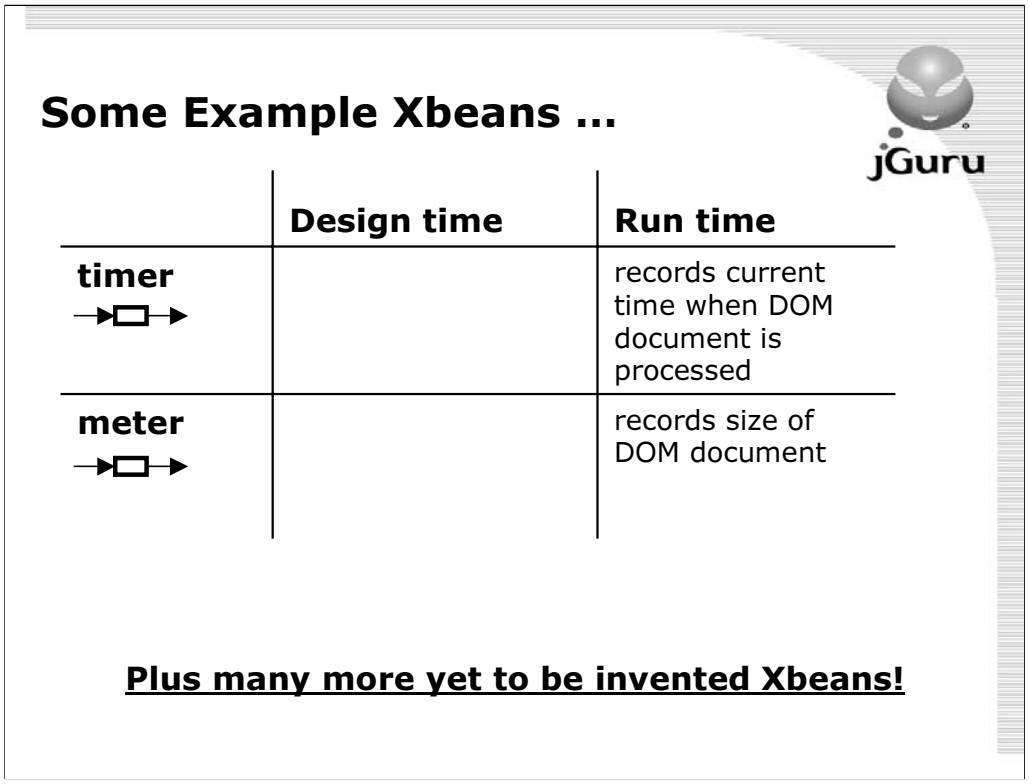
Here is a summary of some useful Xbeans. Once again, recall that the properties set at design time can also be set at run time.

Some Example Xbeans …

	Design time	Run time
viewer 	presentation properties	visualizes the incoming DOM document
parser 	the XML source	produces a DOM document
writer 	the output	produces XML text from a DOM document

Some Example Xbeans …

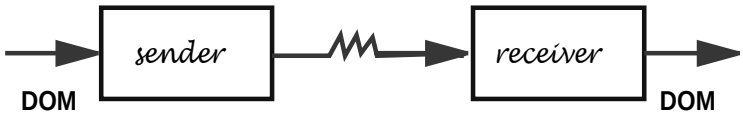
	Design time	Run time
parallelizer 	whether to copy or share a reference to outgoing DOM	creates multiple threads, shares or copies DOM
synchronizer 	criteria for choosing source DOM	receives multiple DOM documents and passes on one
logger 	where output should go	produces a log record for each incoming DOM document



These are simply examples. Lots of generic and not so generic Xbeans can exist.

XML Over the Wire

Sender and receiver Xbeans *abstract the communication of XML* "over the wire".



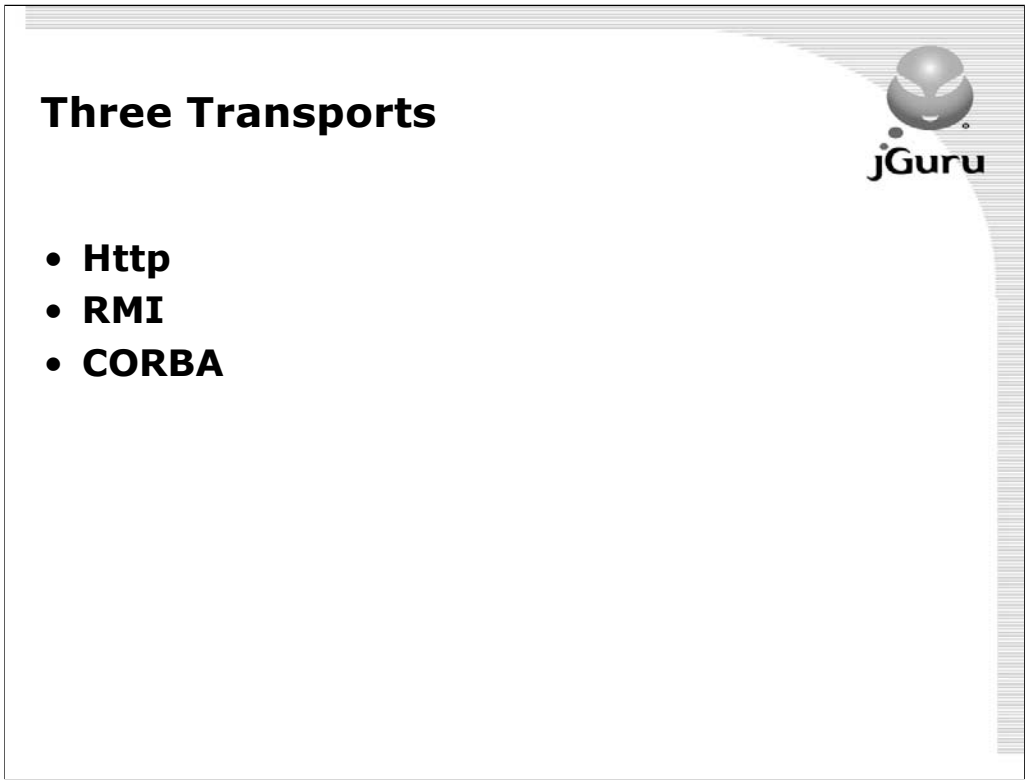
We now turn to looking at the sender and receiver Xbeans in depth.

The sender and the receiver abstract the communication of XML "over the wire."

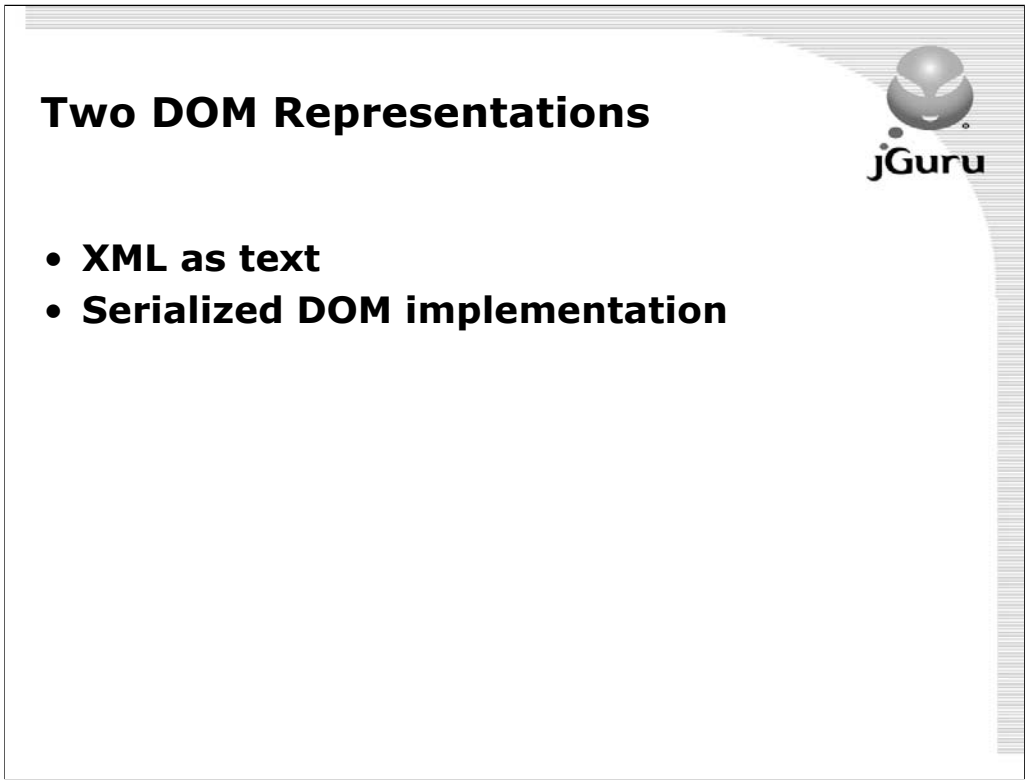
We now look at three different approaches to implementing the sender and the receiver. After

presenting the three approaches, we analyze the performance of each approach.

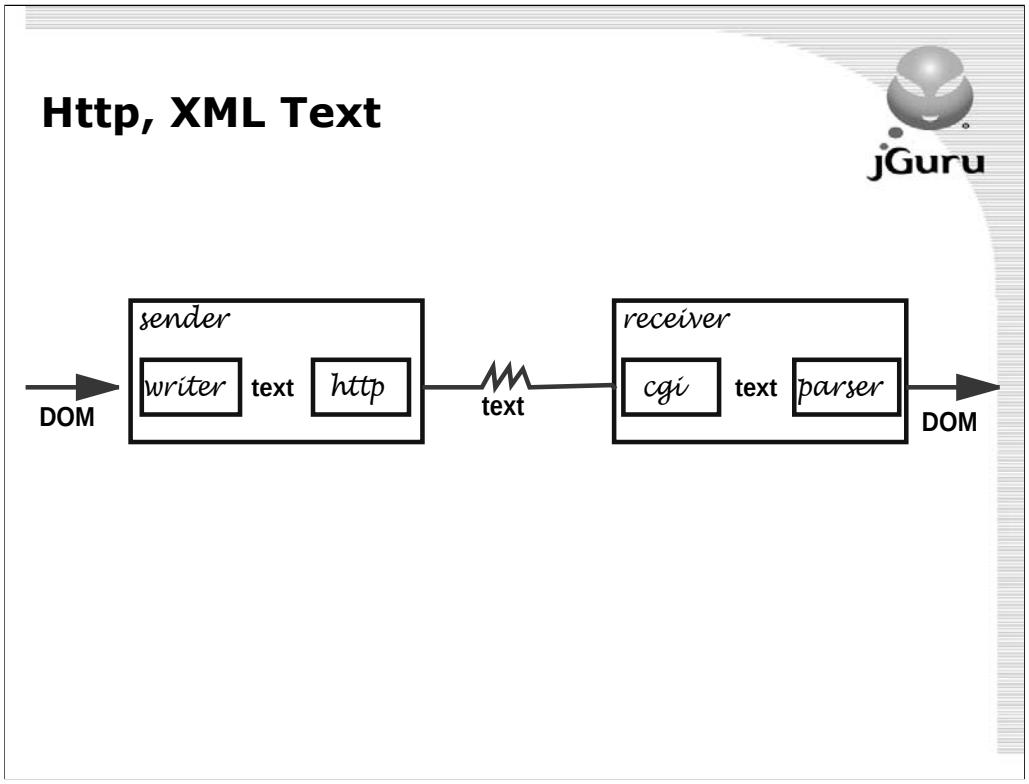
The code for these approaches can be found in three different implementations of the sender and receiver Xbeans. Xbeans are available at http://www.xbeans.org . Also, the February, 2000 JavaWorld issue gives the code.



We compare three different transports. The standard http protocol, Java Remote Method Invocation (RMI) and the OMG's Common Object Request Broker Architecture (CORBA).



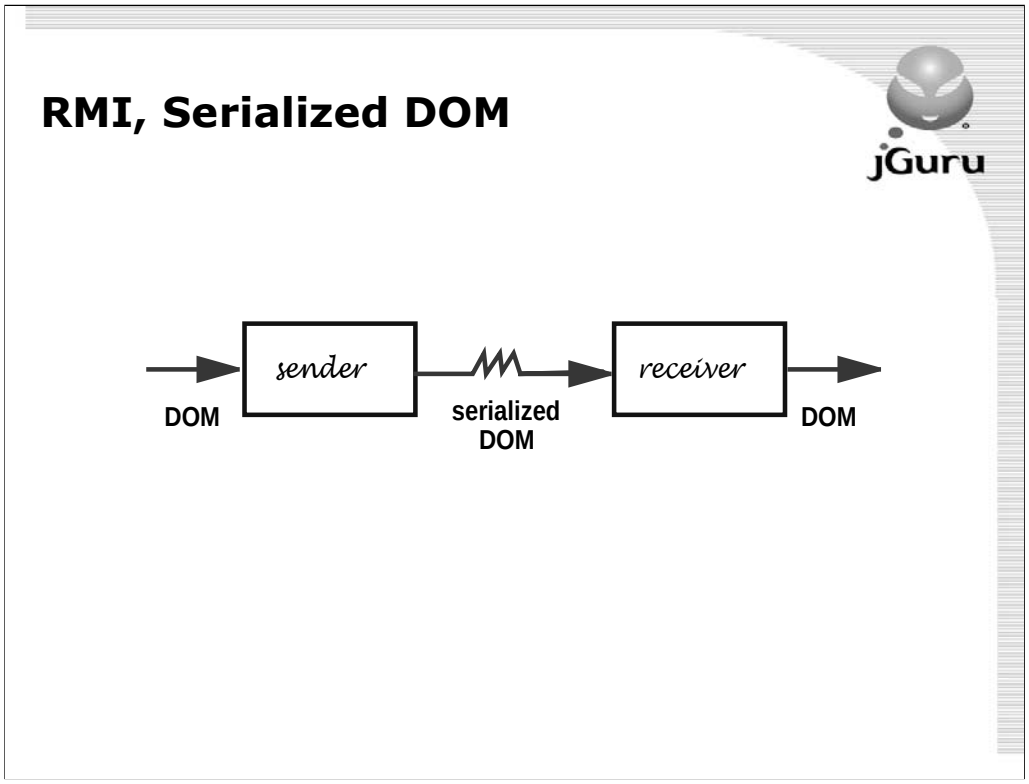
We compare two different on-the-wire representations of the DOM. The first is to use XML's human readable text format. The second is to use Java serialization of the DOM for DOM implementations that support serialization.



The first approach is to simply send the XML to the web server on a remote machine as text. The sender must convert the

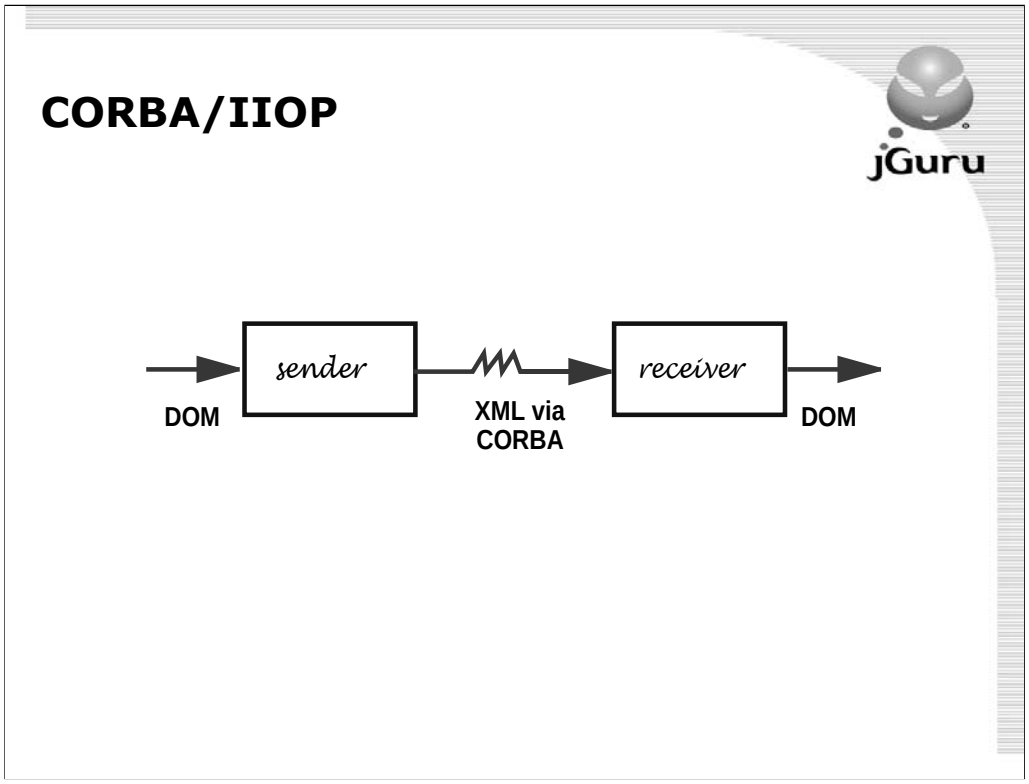
DOM representation of the XML into text and communicate the text to the receiver. Then, the receiver must convert the text

back to the DOM representation.



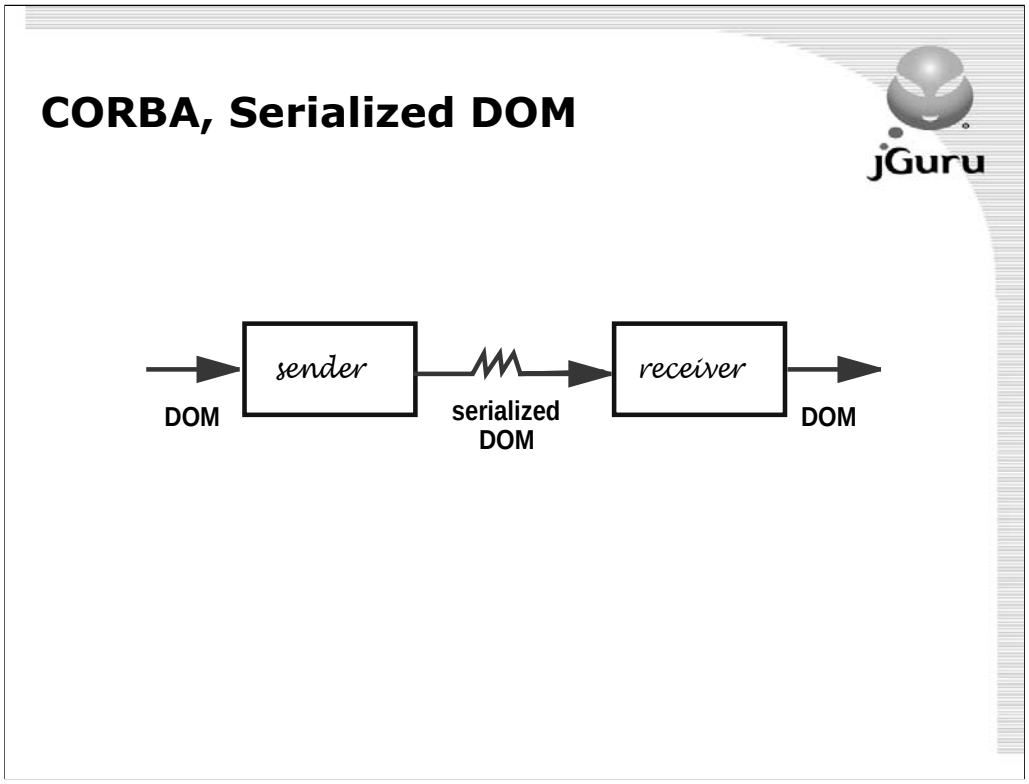
In the second approach we will use Java RMI and serialization of the DOM to transfer the DOM from the sender to the

receiver. The code for this is very natural. The sender simply invokes an operation on the receiver. Java RMI automatically serializes the DOM document, passes it to the receiver and deserializes it in to a DOM document.



The third approach uses CORBA/IIOP as a transport. The Object Management Group currently has a Request for Proposal

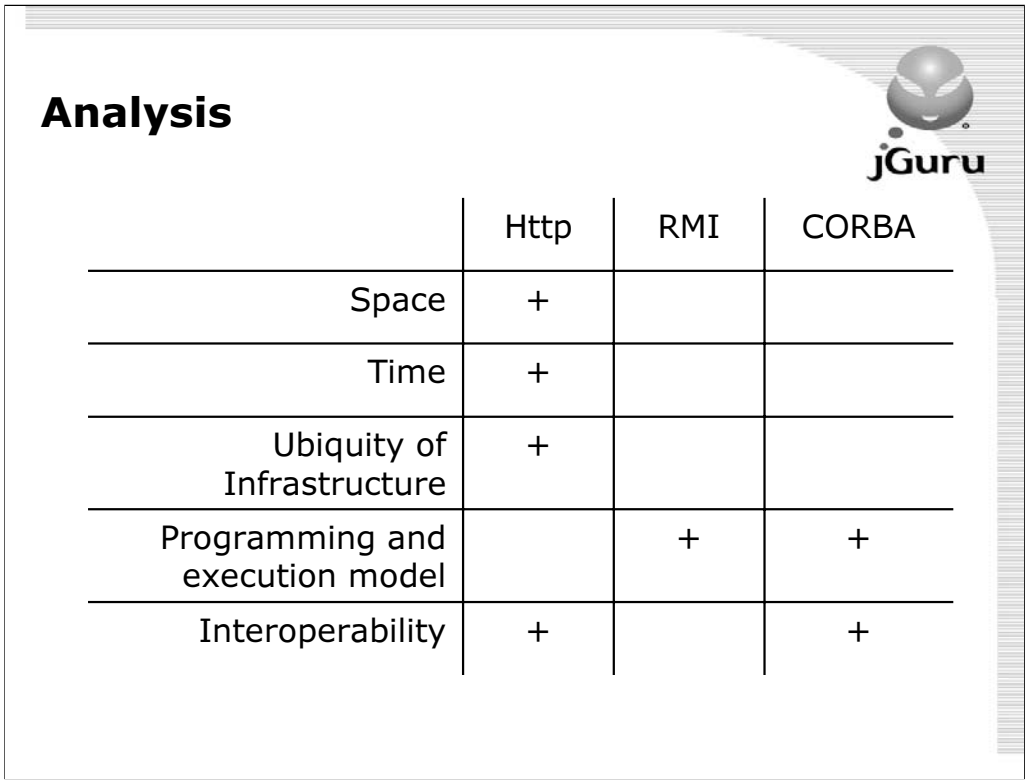
for extending IDL to include an XML data type. Thus, in the future CORBA products will be able to transmit XML natively.



Since CORBA products available today cannot represent XML natively, the code presented here uses Java serialization of the

DOM and communicates it to the receiver as a sequence of octets. It is very similar to the Java RMI approach. The only

difference is that the distributed object infrastructure is provided by CORBA systems.



Space and Time

Surprisingly, the performance results presented later indicate that a textual representation of XML is far a more

efficient representation than a serialized DOM representation. Also, the time required to externalize a DOM

representation and reparse the textual form is cheaper than the direct Java serialization and deserialization of the

DOM. See the performance results below for more details.

Ubiquity of infrastructure

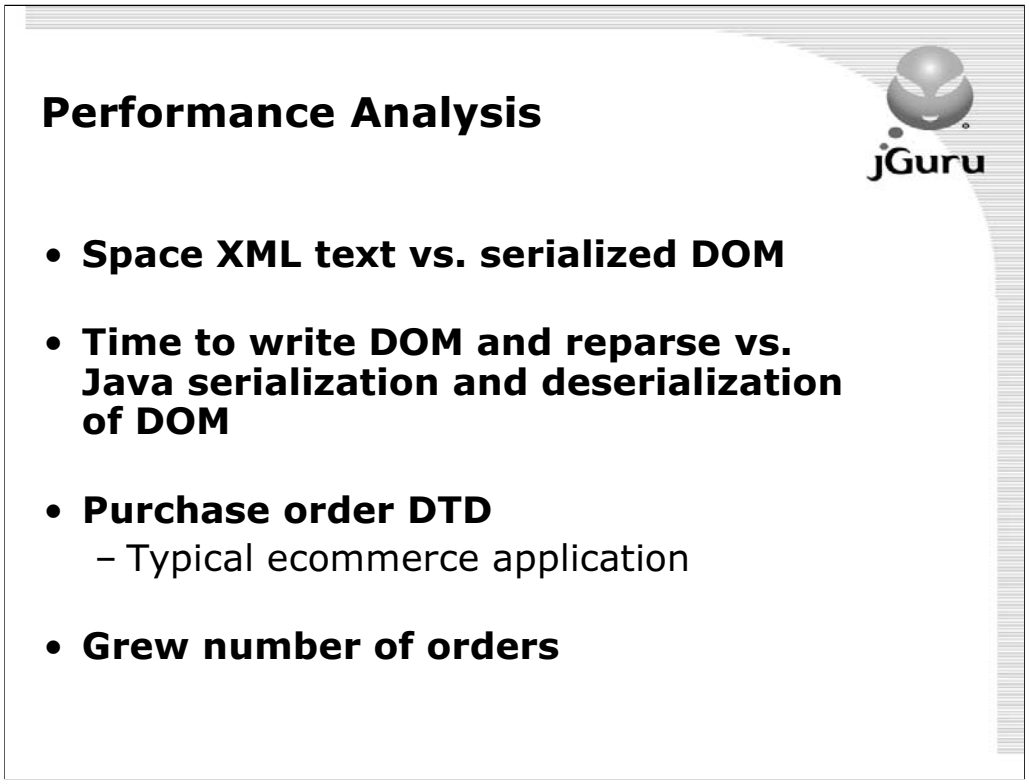
The advantage of the standard web server approach is that the infrastructure to support it is ubiquitous. The cgi

script can be executed from most web servers. Furthermore, the receiver is easily identified using an URL. The

naming infrastructure is already in place. The RMI solution, on the other hand, requires the RMI registry.

The CORBA solution requires the installation of an CORBA Object Request Broker on the server. Furthermore,

the CORBA implementation of the sender uses a URL naming scheme



We did space and execution time performance tests.

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.

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 5555 5555 5555</credit_card>
      </billing_info>
      <shipping_info>
```

Here's the purchase order document we used.

Note that the document can contain one or more orders. We simply grew the size of the XML document by including the same order n times.

Purchase Order …

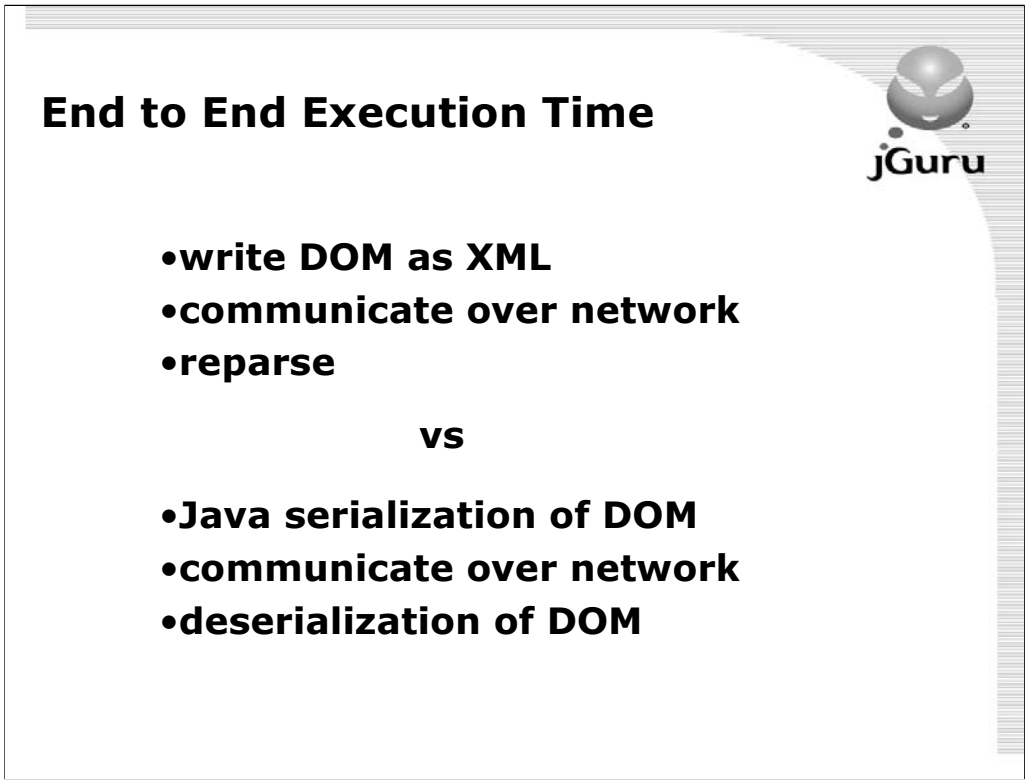

```
        <name>
          <given>John</given>
          <family>Doe</family>
        </name>
        <address>
          <street>555 Main Street</street>
          <city>Mill Valley</city>
          <state>California</state>
          <zipcode>94520</zipcode>
          <country>USA</country>
          <phone>707 555-1000</phone>
        </address>
      </shipping_info>
    </header>
    <item>
      <product_id>5555555</product_id>
      <product_name>Widget</product_name>
      <quantity>100</quantity>
      <price>.25</price>
    </item>
    <total>25.00</total>
  </order>
</orders>
```

Here’s the rest of the purchase order document.

Space Results

Number of orders	Bytes to represent as XML text	Bytes to represent as 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	9,6385	552,570	5.7
500	481,585	2,755,770	5.7

The Java serialization of the IBM DOM implementation was very expensive. When considering space, simple XML text is a better strategy. Is this a time space tradeoff? We do have to pay the expense of externalizing the DOM document to XML text and reparsing the XML text to the DOM.



We are interested in end-to-end execution between the sender and the receiver. Obviously such timings are going to differ greatly depending on the network communication costs. Let's assume that the communication cost is the same for both approaches for any given network.

Timing Results

Number of orders	Milliseconds to write and reparse DOM	Milliseconds to serialize & 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

We give the time it takes to externalize a DOM representation to XML text and then reparse it into a DOM, as well as 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 is running 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.

Space Results with GZIP Compression

Number of orders	Bytes to represent as XML text	Bytes to represent as serialized DOM	Ratio (serialized / text)
1	(1,048) 384	(7,278) 1,710	4.5
5	(4,900) 425	(29,310) 4,468	10.5
10	(9,715) 468	(56,850) 7,964	17.0
100	(96,385) 1075	(552,570) 70,102	65.2
500	(481,585) 3488	(2,755,770) 344,094	98.7

What if we compress each representation using Java's GZIPOutputStream. Both representations are much improved. The blue numbers are the uncompressed size. The black numbers are the compressed size. The compressed textual representation improves dramatically as we increase the number of orders. It probably is not this good because recall that we grew our document by including the same order over and over.

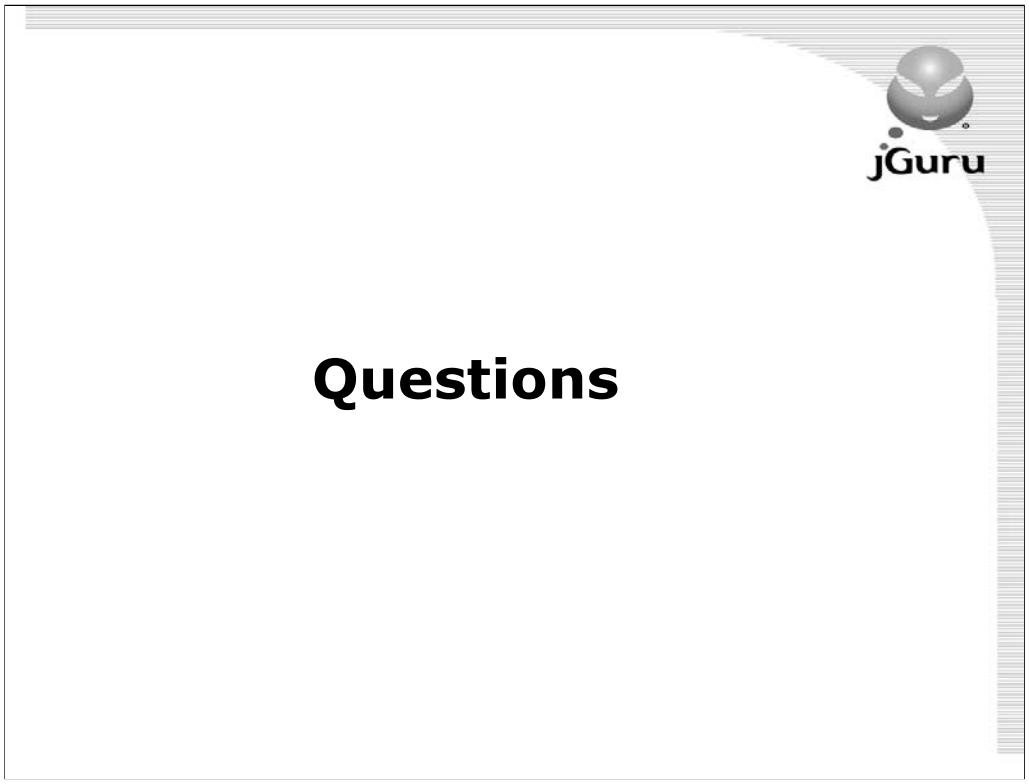
Xbeans.org

- **Open source project**
- **First release available:**
  - Non-atomic sender/receiver for http, rmi, corba
  - timer
  - memory meter
- **Next release of Prof. Volker Turau's DB2xml will be an Xbean – data access**
- **Looking for contributors:**

`www.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.


jGuru
Questions