

1

Overview of the XMLBus

The XMLBus is an XML-based development framework and collection of technologies for building and exposing *Web Services*. The XMLBus enables business-to-business (B2B) transactions and application integration across the Internet, using the existing Internet/Intranet infrastructure available in enterprises.

This chapter briefly describes what XMLBus is for, how it works, the components it includes, and how to use it to build a Web Service. The following topics are covered:

- [“Application-to-Application in a Browserless Web”](#)
- [“How XMLBus Works”](#)
- [“Understanding the Web Services Container”](#)
- [“The XMLBus Components and Tools”](#)
- [“Overview of Building Your Own Web Service”](#)

Application-to-Application in a Browserless Web

The Internet has provided great freedom to do business, but the current model is limited to two tiers: client browsers and servers. Businesses need their Internet clients to access information on the servers, but they have been limited to using Web browsers or building HTML parsers to access server-based functionality. Businesses require the convenience of the Internet to make applications interact directly in a browserless Web.

1 Overview of the XMLBus

Figure 1 shows how Web Services are the infrastructure that make the browserless Web happen.

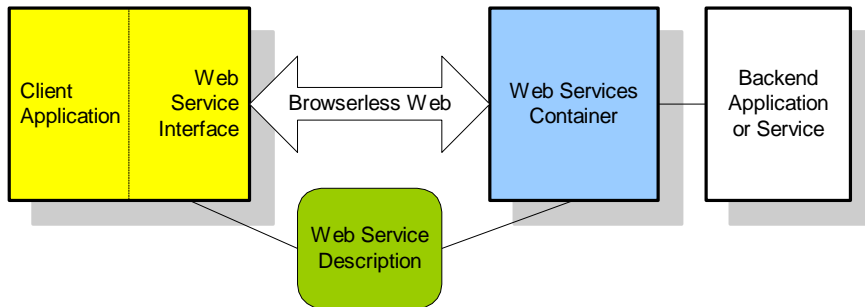


Figure 1: *Web Services Enable a Browserless Web*

A Web Service description gives clients and servers a consistent description of a service's location, operations, and how to communicate with the service. A client application uses an interface that is based on information in the Web Service description to communicate with a Web Services Container. A Web Services Container manages the requests and responses for the deployed Web Services on a particular server.

How XMLBus Works

The XMLBus provides the framework and tools to automatically convert an existing server application into a Web Service, deploy the newly created service, and manage interactions with it. The XMLBus also provides the tools to quickly browse the Internet for other Web Services and then easily use them. The XMLBus makes all this happen without additional programming.

The XMLBus uses Web Service standards for smooth interoperability. These standards include the following:

- Standards and protocols, including the Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Hypertext Transfer Protocol (HTTP), and others give consistent Internet data transfer with other built-in features.

- The Web Services Description Language (WSDL) describes Web Services, the protocol to communicate with them, and their Internet location. A Web Service's WSDL file is a client-server contract that defines the application-to-application communication.
- The Universal Description, Discovery and Integration (UDDI) protocol is for easily advertising descriptions of Web Services and provides businesses with a way to discover Web Services offered anywhere in the world.

Figure 2 shows the architecture of XMLBus and where each of the Web Service technologies play a part.

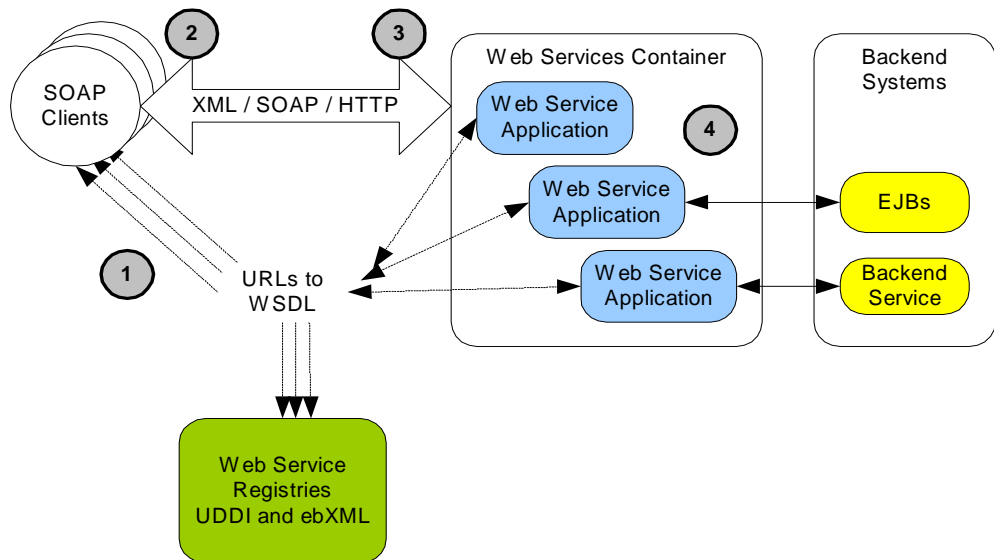


Figure 2: *The XMLBus Architecture*

If you assume for this discussion that some Web Service is already deployed and running, here is how the XMLBus works:

1 Overview of the XMLBus

1. The client must find out where the Web Service is and how to interact with it. The Web Service's WSDL file contains this information and describes the Web Service interface. The client needs a well-known URL to the WSDL file in order to use the Web Service.

The WSDL's URL might be obtained from the Web Service provider, using traditional correspondence such as E-mail, or the URL might be obtained from a global UDDI repository, if the provider has registered the Web Service in UDDI.

2. A client invokes a method on a Web Service using the XML-based SOAP and HTTP protocols.

The XMLBus provides a convenient *Web Services Test Client* that you can easily use to try out a Web Service without doing your own programming. Java client stubs are also provided that automatically convert Java method invocations into Web Service requests. Programmers wishing to use other languages can build clients that adhere to the standard WSDL generated by XMLBus.

3. Information in the SOAP message directs the HTTP post to the appropriate server-side XMLBus *Web Services Container*.

The Web Services Container has a SOAP listener that validates the SOAP message against the corresponding XML schemas, as defined in the WSDL file, and then unmarshals the SOAP message.

Within the Web Services Container, dispatchers invoke the corresponding Web Service implementation code.

4. Finally, the backend implementation executes and performs whatever the Web Service was designed to do. The results of any method calls are transformed into a SOAP response and returned to the client.

Understanding the Web Services Container

The XMLBus provides a Web Services Container as a first-class environment for deploying and running Web Services. The container is a ubiquitous, accessible, and consistent platform for Web Services.

Ubiquity: The container runs on multiple platforms, including IONA's iPortal Application Server, BEA's Web Logic Server, and IBM's WebSphere Application Server. The container can also run in stand-alone mode without a J2EE application server.

Accessibility: The Web Services Container provides access to service lifecycles and the runtime environment. Management interfaces facilitate service deployment and administration.

Consistency: The Web Services Container consistently isolates application implementation logic from the Web Services infrastructure. The container uses third-party class libraries to manipulate XML, introduces proprietary code, and maintains the runtime state. The Web Services Container isolates implementation code from this environment. Because of this isolation, the environment provided to hosted functionality is consistent across platforms and transports.

The Structure of the Web Services Container

The Web Services Container is the runtime part of the XMLBus, which contains any number of Web Services. [Figure 3](#) shows the inside of a Web Services Container.

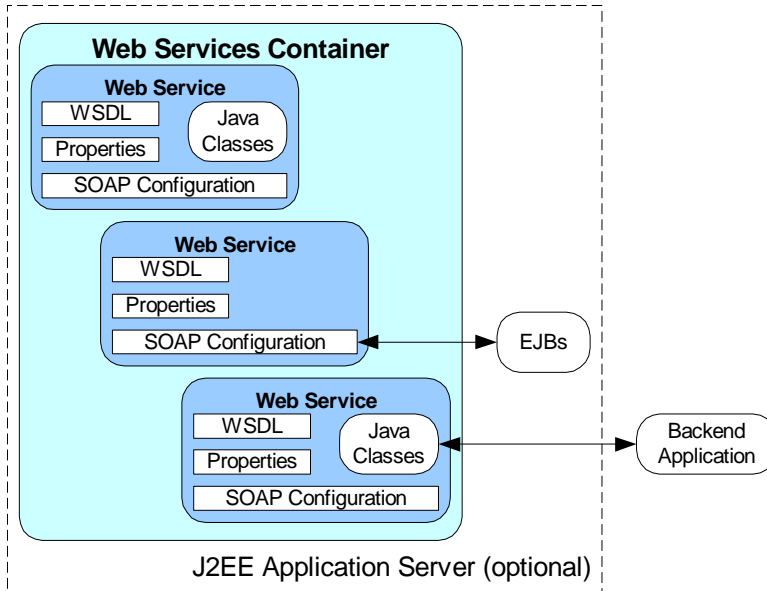


Figure 3: *What's Inside the Web Services Container*

The Web Service implementation can be included in the Web Service. If the Web Service implementation consists of local Java class files, these files and any class dependencies are included in the Web Service. If the Web Service implementation consists of EJBs, the required reference information is incorporated into the SOAP configuration. The Java or EJB implementations can also access any backend applications they need in their usual way.

XARs are XMLBus Archives

An XMLBus Archive (XAR) file physically represents a Web Service and is a single package with all the elements that make a Web Service work. The Web Service Builder produces the XAR files and the Web Services Container uses the XAR files to manage its Web Services. [Figure 4](#) shows the kinds of information that get packaged into a XAR.

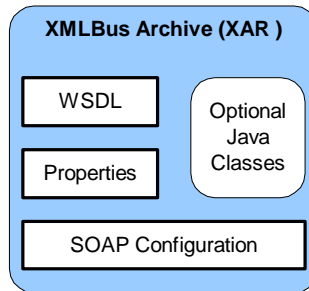


Figure 4: *An XMLBus Archive (XAR)*

WSDL file: A Web Service description language (WSDL) file is a consistent way of describing a Web Service.

Properties file: A properties file contains information about how the Web Service was created. For example, the file contains the URL for the Web Service endpoint, and the class and methods the Web Service supports, among other things.

SOAP configuration file: The SOAP configuration file has information about where implementations are. For example, it describes where EJBs are located outside of the Web Service container in the application server.

Java classes: These are optional implementation files that can be selected when the Web Service is created.

A XAR file can also bundle together more than one Web Service, to create a Web Service application.

Deploying Web Services

The XAR file contains all materials that the Web Services Container needs to launch and run the new Web Service. After the Web Service is encapsulated as a XAR, the Web Service Builder can deploy it directly into a running Web Service Container. The container updates immediately and re-loads any changed classes.

Application Servers and the Container

The Web Services Container currently supports four server platforms:

- Stand-alone
- IONA's iPortal Application Server
- BEA's WebLogic Server
- IBM's WebSphere Application Server

The stand-alone container has no external dependencies and supports only class-based Web Services.

The Web Services Container supports EJB-based applications by using one of the J2EE application server platforms. During installation, the Web Services Container is added to the application server automatically. With this type of installation, the Web Services Container can automatically deploy EJB-based Web Services into application servers.

Although the underlying enterprise applications may change relatively slowly, Web Service interfaces can change rapidly. The container isolates the Web Services from the J2EE application servers, so the Web Services can be deployed, re-deployed, and modified without effecting the underlying application functionality. This isolation means that changes to the Web Service require no application server restarts.

Because all four platforms support persistent deployment, the Web Service container is a new, long-lived enhancement to these server environments.

The XMLBus Components and Tools

The XMLBus includes tools to start using existing Web Services right away, but the real power it provides is in its ability to transform an existing application into a Web Service. Figure 5 shows the components of the XMLBus.

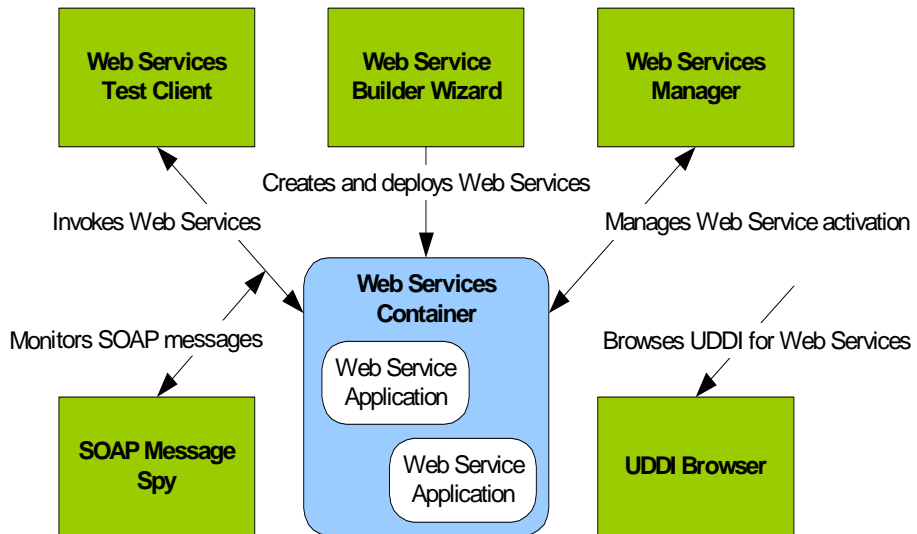


Figure 5: *The Components of XMLBus*

Web Services Container

The *Web Services Container* is the runtime component that provides a bridge between Web Services and existing logic. This message handler listens for SOAP requests and dispatches the requests to the Web Service application that can handle the requests. It also manages the responses, making sure they are correctly formatted and delivered back to the appropriate client.

Web Service Builder

Developers typically use the *Web Service Builder* to create a Web Service from a working application such as a Java or EJB component. This tool generates a XAR file. The XAR file includes a WSDL file (which describes the newly created Web Service) and information the server-side needs to deploy the service. In addition, the wizard can produce a fully functioning, stand-alone client that uses the new Web Service.

Web Services Manager

Server developers use a Web Services Manager tool to administer the Web Services in the Web Service Container. The Web Services Manager displays the Web Services, the WSDL information for each Web Service, and the endpoints on which an implementation is running.

Web Services Test Client

This graphical tool provides a generic client for testing Web Services. The Web Services Test Client uses any Web Service's WSDL and works in conjunction with the SOAP Listener to execute operations of a Web Service. The WSDL can be one generated by the Web Service Builder or any valid WSDL file.

SOAP Message Spy

The SOAP Message Spy lets developers monitor the SOAP request and response messages between clients and servers of Web Services. Developers can also enter a SOAP request directly, send it to a server, and monitor the result.

UDDI Browser

This graphical tool allows you to traverse UDDI repositories for Web Services. These global repositories provide places on the Internet where businesses can advertise or find Web Services.

Overview of Building Your Own Web Service

This guide explains how to use the XMLBus to create and manage Web Services. This includes:

- How to use the tools and what to supply to them
- What components are used where in the development process
- What files and information are created

Web Service development typically involves the following phases:

1. Building a Web Service

This phase uses the Web Service Builder tool, which takes a Java class or EJB and generates a WSDL that describes the application in Web Service terms. While using the Web Service Builder, the developer also provides WSDL binding information and other Web Service information that gets stored along with the WSDL in an XMLBus Archive File, or XAR file. The WSDL generated can be subsequently used by the Web Service Builder to generate client application code that you later use to access the new Web Service. Server skeleton code can also be generated. Finally, the new Web Service can be automatically deployed.

2. Managing a Web Service on the server

The Web Service Container is the runtime component in which the Web Services are deployed. The XAR files generated by the Web Service Builder represent the Web Service applications.

The Web Service Manager tool monitors and controls the Web Services running in the Web Service container.

The SOAP Message Spy tool lets the developer monitor SOAP messages during client and server interactions.

A developer can also run the Web Services Test Client to try out Web Services. This is a JSP-based graphical tool that allowing the user to interact with any deployed and running Web Service.

3. Using a Web Service from clients

In this phase a client developer accesses the Web Service by compiling and running the static or proxy clients generated by the Web Service Builder. The code from these clients can also be incorporated into other client applications.

Summary of Benefits of the XMLBus

A summary of some of the benefits XMLBus provides are:

- Web services can be rapidly implemented and deployed.
- Developers can implement web service business logic using their preferred programming platform including, Java and J2EE.
- Businesses can more readily expose services to their potential partners and customize services based on business needs, without affecting business logic or writing complex, custom code.
- The XMLBus combined with SOAP allows businesses to exchange data directly, without HTML, web servers, or browsers.
- Provides an interoperability platform for .NET and J2EE.