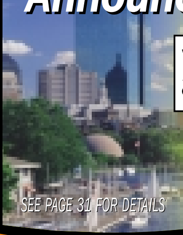
MAKE WEB SERVICE DEVELOPMENT EASIER

January 2003 Volume 3 Issue 1

WebServices JOURNAL

.NET J2EE XML

WSJ2.COM



Announcing...

web services EDGE conference&expo

March 18-20, 2003

Boston, MA

HYNES CONVENTION CENTER

SEE PAGE 31 FOR DETAILS

From the Editor
All I Want For...
by Sean Rhody pg. 3

Commentary
The New Integration Architect: You
by Dave Chappell pg. 4

EAI
Augmenting EAI with Web Services
by Jim Fisher pg. 16

Product Review
J2EE Release 1.4
Reviewed by Joseph A. Mitchko pg. 28

Industry Commentary
Beyond the

WASP
Taking the Sting Out of Web Services Development
The way to deploy a simple Java Web service
PAGE 10

SPECIAL FOCUS ON WEB SERVICES TOOLS, TIPS AND

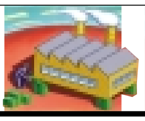
Beyond SOAP: Optimized Web Servi
A more optimal transport with
Putting W
Carlson 30

simple and the complex Aravilli Srinivasa Rao 34

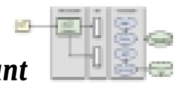
Services Development with PowerBuilder 9 Build Web service clients in both Windows and JSP apps Jinyou Zhu 40

Techniques for Robust, Database-Driven Web Services Reuse of existing assets can create new interoperability Steve Muench 50

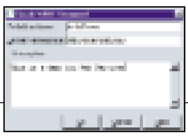
Discover Web Services for Yourself Architect a dynamic discovery with a publicly available service Ron West 54

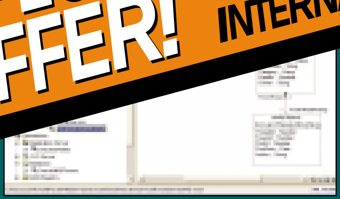


Success Story: Web Services Made Sense! Technology that makes everyday tasks simple, fast, and elegant Lino Tadros 7



SunONE: The Symbiosis of Innovation Building standards-based Java Web services Mike Bellissimo 46





SPECIAL OFFER! WEB SERVICES EDGE 2003 EAST
INTERNATIONAL WEB SERVICES CONFERENCE & EXPO
SAVE $400 IF YOU REGISTER BEFORE FEBRUARY 28, 2003

RETAILERS PLEASE DISPLAY UNTIL FEBRUARY 28, 2003

$6.99US $7.99CAN

SYS-CON MEDIA

Sonic Software

www.sonicsoftware.com/websj

WebServices JOURNAL
.NET J2EE XML

EDITORIAL OFFICES
SYS-CON MEDIA
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9637
WEB SERVICES JOURNAL (ISSN# 1535-6906)
Is published monthly (12 times a year)
By SYS-CON Publications, Inc.
Periodicals postage pending
Montvale, NJ 07645 and additional mailing offices
POSTMASTER: Send address changes to:
WEB SERVICES JOURNAL, SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645

All I Want for...

Written by
Sean Rhody

Author Bio:
Sean Rhody is the editor-in-chief of ***Web Services Journal****. He is a respected industry expert and a consultant with a leading Internet service company.*
SEAN@SYS-CON.COM

It's not unusual for the January issue of a magazine to have a column discussing the past, or predicting the future. This year, I thought we might try something a little different. Rather than reminisce or prognosticate, I thought I'd toss out my list of Web service needs in the form of a holiday wish list.Here goes:

Wish 1: A unified security standard. SAML, WS-Security, I don't much care which, but I want a single standard to prevail, so that we can all get past the "Web services aren't secure" issue and get on with the real work. Let's have the implementers innovate around a single standard instead of dancing around trying to support competing standards, or waiting on the sidelines for the dust to settle.

Wish 2: A better UDDI. UDDI is too much of an all-or-nothing technology. You publish something publicly and the whole world knows. Almost no one wants that type of visibility. That's one of the reasons why people are either using private UDDI registries or foregoing UDDI altogether. Even in private registries, this causes problems when you have a segmented user community. UDDI should work seamlessly with the security standard (see Wish 1) to allow creation of private communities within the same UDDI tree. And don't tell me I can do this with multiple UDDI servers – that's a workaround, not a solution.

Wish 3: A simple definition of Web services that the world can agree on. Even our advisory board has trouble with a single definition. Given the issues with UDDI (see Wish 2), it's hardly surprising that we can't agree on that. Add to that the fact that SOAP is really a transport mechanism that can be replaced with other technologies and things just get too murky to even put forth an opinion. Can't we all just get along?

Most people only get three wishes, but I'm feeling really good about 2003, so I'll throw out a few more for good measure.

Wish 4: OASIS, W3C, and WS-I should merge into a single standards body, preferably run by me. Seriously, the competing standards we deal with are impeding everyone's progress (see Wish 1). A single group, like the JCP, would provide a better path toward innovation without churn.

Wish 5: A cross-platform development tool so that I never have to look at WSDL or UDDI entries again in my life. Honestly, it's like looking at HTML code. I can, but why should I have to unless there's a problem I need to debug? Of course, such a tool has to work with Visual Studio as well as all the Java tools so I know I'm wishing in vain here. But I want something to let me construct WSDL and UDDI entries graphically, and then let the other IDEs generate the starting code structures, rather than developing code first, then WSDL and UDDI would be a gigantic step in the right direction.

Wish 6: An economic upturn. While this is not specific to Web services, as the saying goes "a rising tide lifts all boats." With a return to prosperity, all of the things we're looking for will come at a faster pace, spurred on by revived business needs. And everyone just feels better in a growing economy.

Thanks for listening to my wish list, and for reading ***Web Services Journal***. On behalf of my staff and all of SYS-CON, I'd like to extend my personal best wishes to you and yours for the coming year.

Happy New Year, and good reading.

The New Integration Architect: You

XML technology could be the root of a successful integration project

Dave Chappell

Dave Chappell is vice president and chief technology evangelist for Sonic Software, the leading provider of integration products and services for the real-time enterprise. Dave is co-author of Java Web Services *(O'Reilly & Associates, 2002),* Professional ebXML Foundations *(Wrox, 2001), and* The Java Message Service *(O'Reilly & Associates, 2000), and a frequent contributor to* Web Services Journal.

CHAPPELL@SONICSOFTWARE.COM

According to Gartner, Inc., vice president and research fellow Roy Schulte, "a new form of enterprise service bus (ESB) infrastructure will be running in most major enterprises by 2005." ESBs combine Web services, enterprise messaging, transformation, and routing to provide an integration network that can span global enterprises and encompass potentially thousands of application end points. Application integration is a top priority among CIOs, and as the current IT value center in the enterprise, IT organizations must shift their focus from application development to application integration. The convergence of this IT spending trend, and the availability of standards-based technology and tools for integration, means IT developers are the next integration architects.

While a thorough technical analysis of this ESB infrastructure is beyond the scope of this brief editorial, I would like to look at what many consider the defining characteristic of integration: transformation. As the focus of this month's issue is tools, tips, and tricks, It seemed appropriate to focus on the importance of tools for creating and managing XML and XSLT transformations.

The Role of XSLT Transformation

XML is a common way of representing data that is being routed between applications within an enterprise, and between business partners. However, standards for packaging XML are constantly evolving. We have competing standards in place for representing common business objects, such as billing and ship-to addresses. Business partners may have chosen to implement their own proprietary XML formats. Yes, we live in a time where we already have legacy XML formats that we have to deal with!

It is fairly common practice for an enterprise to use a single common format for packaging up XML data as it travels throughout the enterprise between applications across the ESB, then set up translation services to convert data to and from the target formats. The translation service can be part of an adapter to an application, or can serve as part of a collaboration with an external business partner. The idea is that each transformation service at each end point can be solely responsible for translating data between the common XML format and the target format of the application consuming the data.

XSLT (eXtensible Stylesheet Language for Transformation) is an ideal way of converting XML data from one form to another. Using an XSLT processor, a stylesheet can be applied to a source document to produce one or more output documents. The output document can be another XML document, HTML, or plain text. The XSLT processor can add to, copy, remove, or rearrange the contents of the source document in accordance with the instructions found in the stylesheet. Extensions may be applied to the transformation in the form of Java, JavaScript, VBScript, or any language the XSLT processor supports, to allow more detailed control of the transformation.

The Role of Tools in Creating and Managing XSLT Transformations

You might consider yourself a code-slinger who doesn't need any fancy tool stuff. You've been using vi for years and that's all you'll ever need. You may even have migrated to Notepad. Why go further? For starters – XML is extremely verbose! Well, that's old news, but that in itself makes one yearn for simple pleasures like color coding, bracket matching, and tag completion. That's all well and good, but how do tools apply to XSLT? What are the issues? Why should you care? Here are a few reasons:

- ***XSLT is a declarative language:*** Unnatural for programmers who have been trained in and have been doing procedural programming for years.
- ***XSLT is heavily driven by the source document:*** Makes it very tricky to predict the outcome simply by visually scanning the source document and the stylesheet.
- ***Creating transformations:*** A simple case is a source document that has <zip-code>, the target document needs <postalCode>. A stylesheet transformation can also include more complicated things like extracting multiple line items of a purchase order into separate XML messages. How about being able to just visually place the source and target documents side-by-side, connect the dots, and let a tool generate the XSLT for you?
- ***Modifying transformations:*** What's the round-trip experience of changing an XML document and updating its associated stylesheet? Why should you manage this by hand, when you can automatically generate it with a tool?
- ***Eventually this XML data that is being shipped around needs to be visualized by humans:*** Either for diagnostic purposes or for formal presentation in applications. XSLT is perfect for that job too. Having a tool that allows you to drag and drop elements and attributes from an XML document directly onto an HTML can-

Continued on page 57 ▶

ParaSoft

www.parasoft.com/wsj1

Application integration is an intimidating challenge for any enterprise; the downtime costs of unreliable integration are unthinkably expensive. To keep profits climbing, enterprises are adopting open standards-based messaging, caching and integration solutions.

The SpiritSoft framework integrates your legacy applications into a single JMS-compliant environment. SpiritSoft technology goes beyond JMS to give you a secure, scalable and robust enterprise-integration strategy to provide for your future messaging and Web Services requirements.

Secure your integration foothold … there's no room for error.
Download our white paper www.spiritsoft.com/climber

spiritsoft.
go beyond jms

Web Services Made Sense!

Technology that makes everyday tasks simple, fast, and elegant

It was an exciting project!

Kazoo's CEO, Joseph Cardoza, was asked by Colliers' CIO, Jon Green, to propose a solution for a problem that had been haunting the Colliers IT department for a while.

The problem was "speed." Requesting a commercial real estate report from a computer system should never be a "Project" that requires coffee making and donuts to be consumed while waiting.

The system was ASP based and connected to MS SQL Server 2000 to query commercial real estate properties. The reporting portion of the system was just thrown together!

The agents at Colliers requested that the reports be in PDF format. Well, that's good except getting there from an ASP page is not as easy as advertised.

Adobe Acrobat allows for PDF creations, but for more control over the output, format, and content, Crystal Reports was chosen to control the output because of their ability to construct the output in PDF format. At that point, we felt we were almost there, but not quite.

The COM wrapper for the Crystal engine to be called from the ASP page to do the job was "single threaded": that means if 10 agents hit the site and start printing reports, they will be worked on one at a time, causing a jam and complete dissatisfaction, especially if Agent 005 wants to print all the available properties in California. You get the picture.

We found a solution by developing a Borland Delphi-based COM server that receives the printing requests first and multithreaded them to instances of the Crystal engine.

Did I mention Web services yet? Bear with me.

The solution worked, everybody felt relieved that reports were moving along, no jams, and they got the job done.

The reality of the matter was that we didn't do anything. We just fixed something that was supposed to work right from the start. The real problem was: How could we get Colliers Agents to be more productive on the go?

If I am a customer requesting knowledge of commercial real estate availability from a Colliers agent, the process is for that agent to print a log of available properties in that area and hit the road. What happens when I, the customer, have a question about a specific property that is not on the printout? What happens when I choose to change the location of the search halfway through? Do we have to go back to the office for more printouts? What a waste of time.

The goal was to spice up the current framework to allow for easy access to this information on the road. Plugging in a laptop on the road and hitting the site was not my idea of "spice up."

At the time, I wanted to use the SOAP framework built in Delphi 6 to wrap the COM requests we built into the framework and allow access to the data over HTTP from any platform or device worldwide. It was extremely easy to build this server using Delphi 6, the only commercially available SOAP framework at the time. All the SQL queries were routed straight back to the transaction server already built using COM and MTS and the responses were all driven back to the requester using SOAP (XML over HTTP).

Portable Data Access

"Okay, now that we have a solution, how are we going to apply it for data access on the road?"

We chose the PocketPC route; it was the best choice for wireless development. We bought several iPAQ 3650s, embedded VC++ for PocketPC and MS SQL for CE and started playing. We decided to use the PocketSOAP framework (www.pocketsoap.com) to provide the client code on the devices for accessing the Delphi 6 Web service we had just created. It was amazing how fast everything was put together, and it worked!

One final problem we were facing was the choice of hardware for wireless access.

When we started this project, Ricochet was still alive and was our first choice; when they went under, we had a larger problem on our hands because of two major issues:

1. We needed something that worked as well as Ricochet, speedwise, because SOAP was not the fastest animal on the block. It was still TEXT over HTTP.
2. It needed to be financially equivalent to operate (or even better) to get the approval for nationwide use of the technology.

Well, first there was nothing even close to Ricochet. The wireless modems that we experimented with were at

Author Bio

President and CTO of Kazoo Software, Inc., Lino Tadros is a frequent speaker at software development conferences throughout the world and a former software engineer on the Delphi and C++Builder development teams at Borland Software Corporation. He is recognized internationally for his expertise in the areas of COM, XML, SOAP, and Internet development. Lino is a Borland Certified Developer and Trainer on Delphi, Kylix, C++Builder, and JBuilder. He was honored with the prestigious "Trainer of Year 2002" award by Borland.

About Kazoo

Founded in 2000, Kazoo is considered one of the fastest growing companies in Silicon Valley. Using its exclusive and proven consulting process, Kazoo utilizes a wide range of talents that come together to develop customized solutions that fit customers' needs. Kazoo is your partner for powerful business solutions that help your organization increase efficiency, productivity, and profitability.

least 10 times slower, unreliable, and extremely expensive to operate monthly with heavy usage.

Then we started to consider two different approaches: wireless 802.11b technology for internal use in all Colliers offices via an Access Point and Sierra Wireless AirCard 555 for external usage. It worked great!

As part of a pilot program, agents were able to retrieve property information and query their data wirelessly anywhere in the Bay Area.

Return on Investment

But we still had a major question: "How can Colliers make money out of Web services?" This is a question a lot of companies have asked, and they've fallen short of finding a business case where Web services can generate revenue.

That idea started the second phase of our project. Jon had explained to Joe and I the essence of collaboration between companies in the commercial real estate business. It was definitely different from regular real estate. Visiting company XYZ on the Web and requesting views of properties in special areas renders only properties handled, owned, or contracted by company XYZ. Bearing in mind that Colliers International is the biggest commercial real estate company in the world, a lot of other commercial real estate companies would really find it advantageous to tap into a properties inventory like Colliers'.

The original Web service was modified to allow logging, authenticating, and charging of data access to the requesting party. It was based on the same model as a regular Colliers agent, except a flag was introduced in the database to identify the agent as a competitor, which linked in another table that identified the charge per listing.

So, a potential buyer visits www.xyz.com (commercial real estate company) and requests a search of properties of a minimum of 10,000 square acres in the Alameda area. XYZ only has two listings in that area but has a contract with Colliers to access their inventory. A Web service call is made into the Colliers system to authenticate and retrieve the information. If the contract with Colliers states, for example, that each listing will cost 10 cents, the SOAP dataset returned is counted, logged, and saved.

A Linux Mirror

Just before we released the pilot program for the new system, Borland Software Corporation announced the release of Kylix 2.0, a Linux-based RAD development tool identical to Delphi. Knowing the power of Linux on the server side and its cost effectiveness, I was given the green light to experiment with mirroring the Web service solution under Kylix 2.0 for Apache hosting.

I would have liked to mention that we worked hard on this task or even contributed much to this effort, but the truth is that Kylix 2.0 did all the work. It took two hours to convert the SOAP server and that's because we had to mirror the MS SQL Server to Interbase.

Summary

It was a pleasant conversion, which almost never happens in real life. Kazoo Software continues to support Colliers' efforts to make technology a tool to make everydays tasks simple, fast, and elegant.

With the Delphi and Kylix solutions developed by Kazoo, Colliers supports its internal team with state-of-the-art applications designed for speed, flexibility, and performance.

WebServices JOURNAL
.NET J2EE XML

PRESIDENT AND CEO
Fuat A. Kircaali fuat@sys-con.com

COO / CFO
Mark Harabedian mark@sys-con.com

VP, BUSINESS DEVELOPMENT
Grisha Davida grisha@sys-con.com

ADVERTISING
SENIOR VP, SALES & MARKETING
Carmen Gonzalez carmen@sys-con.com
VP, SALES & MARKETING
Miles Silverman miles@sys-con.com
ADVERTISING DIRECTOR
Robyn Forma robyn@sys-con.com
ADVERTISING ACCOUNT MANAGER
Megan Ring-Mussa megan@sys-con.com
ASSOCIATE SALES MANAGERS
Carrie Gebert carrie@sys-con.com
Alisa Catalano alisa@sys-con.com
Kristin Kuhnle kristin@sys-con.com

SYS-CON EVENTS
VP, EVENTS
Grisha Davida grisha@sys-con.com
REGIONAL SALES MANAGERS, EXHIBITS
Michael Pesick michael@sys-con.com
Richard Anderson richard@sys-con.com
CONFERENCE MANAGER
Michael Lynch mike@sys-con.com

CUSTOMER RELATIONS / JDJ STORE
CUSTOMER SERVICE REPRESENTATIVE
Margie Downs margie@sys-con.com
JDJ STORE MANAGER
Rachel McGouran rachel@sys-con.com

SYS-CON.COM
VICE PRESIDENT, INFORMATION SYSTEMS
Robert Diamond robert@sys-con.com
WEB DESIGNERS
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
ONLINE EDITOR
Lin Goetz lin@sys-con.com

ACCOUNTING
FINANCIAL ANALYST
Joan LaRose joan@sys-con.com
ACCOUNTS RECEIVABLE
Kerri Von Achen kerri@sys-con.com
ACCOUNTING PAYABLE
Betty White betty@sys-con.com

SUBSCRIPTIONS
SUBSCRIBE@SYS-CON.COM
1-888-303-5282
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
COVER PRICE: $6.99/ISSUE
DOMESTIC: $69.99/YR (12 ISSUES)
CANADA/MEXICO: $89.99/YR
ALL OTHER COUNTRIES: $99.99/YR
(U.S. BANKS OR MONEY ORDERS)

SYS-CON MEDIA

Sitraka

(now part of Quest Software)

www.sitraka.com/jclass/ws

Written by Kyle Gabhart

WASP: Taking the Sting Out of Web Services Development

The way to deploy a simple Java Web service

In the October issue of *Web Services Journal* (Vol. 2, issue 10), I wrote an article on how to assemble a free C# .NET development environment by combining Eclipse, the Java 2 platform, Microsoft's .NET SDK, and a C# Eclipse plug-in from Improve Technologies. This time, I'm going to extend the Eclipse workbench to provide a Java Web services development environment by adding a free plug-in from Systinet (www.systinet.com).

Java Web Services Development with Eclipse

By way of review, IBM released the Eclipse project (www.eclipse.org) to the open source community in November 2001. Eclipse is an open source platform that provides a core, extensible infrastructure for building software development environments. It provides a basic user interface, and an open API that supports the extension of the product through its plug-in mechanism.

WASP Server for Java is a complete platform for development, deployment, and management of Web service–based applications. A set of command-line tools is provided to allow developers to build Web service–based applications and application clients in Java. For those that prefer a friendlier interface WASP Developer is available as a self-contained module that plugs into your IDE (Sun ONE Studio/Forte/NetBeans, JBuilder, Eclipse/WSAD). It enables developers to develop, test, deploy, and manage Web service–based applications via the standard features and mechanisms available with their IDE. WASP Developer is built on top of WASP Server for Java, containing a complete WASP Server for Java installation, and using its interfaces to accomplish required tasks.

Thanks to Eclipse's plug-in mechanism, it's possible for us to extend it to produce a free Java Web services development, testing, and deployment environment. By combining Eclipse (open source IDE) with the Java platform (free) and the Web services plug-in from Systinet (free developer license), you have a comprehensive Java Web services application development platform that costs no money, and uses less than 150MB of hard disk space!

This article will walk you through the process of setting up this environment on a Windows 2000 machine (the recommended platform) and deploying a simple Java Web service and a client for that service. This combination of tools will also work on Windows 98/ME/NT/XP, Linux, Solaris, QNX, AIX, and HP-UX.

Environment Setup

Before you start, check Table 1 to be sure that your system has the appropriate resources for this configuration.

Although the minimum and recommended values in Table 1 are accurate, you'll be

AUTHOR BIO:

Kyle Gabhart is a senior mentor for LearningPatterns, a dynamic knowledge company providing world-class mentoring, training, and consulting to clients all over the world. Kyle is a popular national speaker and a prolific writer, with more than a dozen technical articles and books to his name. You can find Kyle on the web at www.gabhart.com

KYLE@GABHART.COM.

TABLE 1: System Requirements

	Minimum Value	Recommended Value
Processor Speed	P 133MHZ	PII 266MHZ
RAM	128MB	256MB
Hard Drive Space	160MB	200MB (with 10% of disk free)

much happier if you have more horsepower available. I'm running this configuration on a P4 1.6 GHZ Dell Inspiron with 512 MB and plenty of free HD space.

Assuming that your system has the appropriate resources, you will need to download three components:

1. ***Eclipse 2.0 SDK*** (www.eclipse.org/downloads/index.php): Select the appropriate OS.
2. ***Java Runtime Environment 1.3.1 or 1.4.x*** (http://java.sun.com/j2se/downloads.html): The SDK is considerably larger than the JRE. Since the Eclipse SDK has a built-in Java compiler, you will still be able to do Java development without downloading the Java SDK.
3. ***WASP Developer for Eclipse*** (www.systinet.com/products/wasp_developer/download/license-ec40): You'll need to register and then log in to download this software. It's free for commercial development or noncommercial development and production deployment. Systinet also offers a "Single-CPU Production License" for its WASP Server product.

Once you have downloaded the three files, you can install the software. To install Eclipse, simply extract the zip file in the parent directory where you would like to store Eclipse (it will create a new directory called 'Eclipse' to store the software). To install the JRE or JDK, double-click on the self-install executable. A wizard will walk you through the necessary steps. At this point, you have the basic Eclipse platform installed (the Eclipse SDK comes with a suite of Java development tools and plug-in development tools). We'll look at installing the WASP Developer plug-in in just a moment.

Eclipse Review

The Eclipse IDE is divided into two basic levels: the Workspace and the Workbench:

- ***Workspace:*** Files, packages, projects, and source control connections
- ***Workbench:*** Editors, views, and perspectives

This division is not readily apparent, and is not labeled within the IDE, but it is an important distinction to understand when working with Eclipse or an Eclipse-based tool.

The *Workspace* is very team-centric, focusing on projects and file resources, as well as providing integrated source control capabilities for these resources. The Workspace also consists of the Workbench elements that are currently active (explained below). When you close Eclipse, it saves the current state of the local Workspace, allowing developers to pick right back up where they left off when Eclipse is restarted.

The *Workbench* consists of the visual artifacts that sit atop the Workspace resources and provide distinct views and role-based perspectives of the underlying projects and resources. The user interface paradigm is based on editors, views, and perspectives. An editor is a component that allows a developer to interact with and modify the contents of a file (source code, XML file, properties file, etc.). Views provide metadata about the currently selected resource. This data is pertinent to, but not directly related to, the actual contents of the resource (the contents would be accessed via an editor). Finally, a perspective represents a configuration of related editors and views as well as customized menu options, and compile settings. When we install the WASP plug-in in the next section, two new perspectives will be added to Eclipse: WASP Management and Web Service.

Installing the WASP Developer Plug-in

As of the writing of this article, the latest version of WASP Developer is 4.0.1, and the latest version of Eclipse is 2.0.2. By the time you read this article, these versions will probably have been updated. Consequently, the installation procedure may have changed. You should verify the installation procedure online (http://dev.systinet.com/ documentation/index –> "WASP Developer for Eclipse"–>"Installation"). Assuming WASP Developer 4.0.1, Eclipse 2.0.2, and JDK 1.4, the installation procedure is as follows:

1. ***Unzip the WASP zip file*** into the parent directory for Eclipse (the WASP zip contains a folder named Eclipse, and subfolders that all assume the Eclipse 2.0.x directory structure).
2. ***Create a script*** (or batch file) to invoke Eclipse with. The script requires only one line, and it passes important parameters into the tool that allow WASP to use some important security capabilities built into JRE 1.4 (JAAS and JCE). You can see the script contents in Listing 1.
3. ***Launch Eclipse*** using this new startup script, rather than the usual executable (Eclipse.exe).

These simple instructions work only if you have JRE 1.4, which has JAAS, JSSE, and JCE embedded. If you have an older JDK,

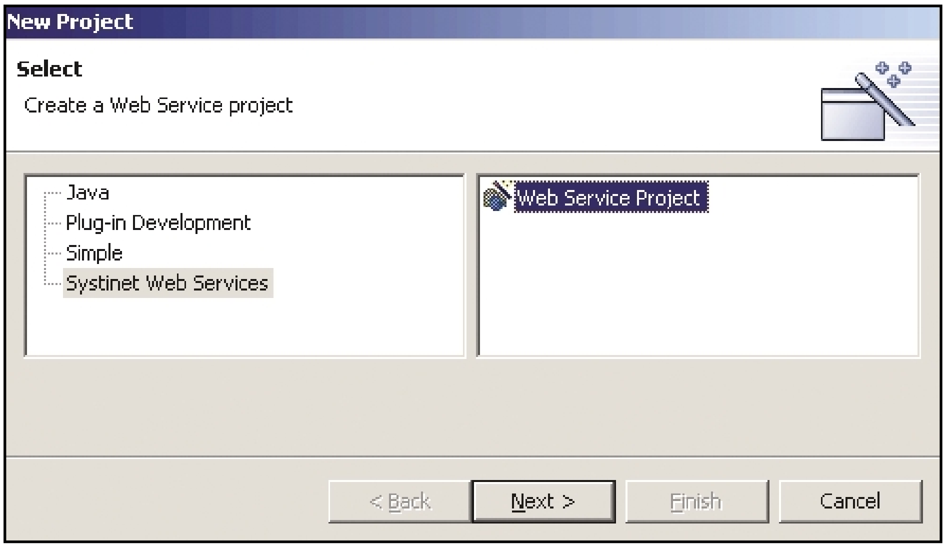


FIGURE 1 Eclipse IDE

the process is as follows:

1. ***Unzip the WASP zip file*** into the parent directory for Eclipse (the WASP zip contains a folder named Eclipse, and subfolders that all assume the Eclipse 2.0.x directory structure).
2. ***In the Eclipse home directory***, rename startup.jar to startup.jar.old. Next, rename wd-startup.jar to startup.jar.
3. ***Open the ECLIPSE_HOME/plugins/com.systinet.wasp_4.0.0/lib folder***. Copy the file named security-ng.jar from this directory into the JAVA_HOME/jre/lib/ext directory.
4. ***Launch Eclipse*** using the standard executable, Eclipse.exe.
5. ***If you have JRE 1.3.1*** and you want to utilize the security features built into WASP, you'll need to follow additional instructions included in the online documentation under the heading "Installing Java Security."

The second method will work for both JRE 1.3.1 and 1.4.x. I prefer the first method, as it requires that no actual changes be made to any JAR files. The second version will work fine until you decide to install an updated version of Eclipse. Then the startup.jar file could be overwritten with a pure Eclipse version, rendering the Systinet-specific launcher files unavailable. Now that you know that, however, you can be sure to save an archived copy of the WASP Developer startup JAR file in a safe place in case you need to reapply it after an upgrade.

Once you've opened the WASP zip file, you can access a local copy of the documentation if you prefer. Launch the following HTML file: ECLIPSE_HOME/plugins/com.systinet.waspdeveloper.eclipse.help_4.0.1/index0.html. This is the index to the WASP Developer for Eclipse documentation. It provides detailed installation instructions, as well as tutorials, a developer's guide, and links to additional resources.

WASP Developer Features

WASP Developer 4.0.1 comes with an impressive list of features, including:

- ***Exposing any Java class as a Web service:*** Develop Java classes in Eclipse, or import existing ones and deploy them as Web services by right-clicking on the class name and selecting one of two Web service creation wizards.
- ***Automatic generation of client-side stubs:*** Building a client to access a local service in development or a remote service in production is easy. A simple wizard walks you through the process of generating the client-side stubs and creates a client Java class for you to complete.
- ***Map between Java and WSDL:*** You can produce a WSDL document for any service, and generate skeleton Java code from a WSDL document.
- ***Inspect SOAP traffic:*** SOAPSpy allows you to inspect the raw SOAP traffic being exchanged between client and service. It is even possible to directly manipulate the SOAP message and resend it for testing purposes.
- ***Full integration with UDDI:*** WASP Developer includes wizards to assist in publishing your Web services to a UDDI registry and getting information back from a UDDI registry to create Web services and their clients, or just to generate the WSDL for your project. No knowledge of UDDI is required.
- ***Extensive Web services security capabilities:*** You can create Web services using authentication, authorization, and message integrity checks. WASP Developer ships with support for three security providers (WASP Security API for SSL, GSS-API/SPKM and GSS-API/Kerberos, and HTTP Basic and Digest Authentication). It also supports the utilization of other security provider technologies via pluggable JAAS modules.
- ***Local and remote debugging of services:*** Eclipse comes with built-in debugging capabilities for local applications, including breakpoints and access to the JVM stack. WASP Developer for Eclipse expands on these basic capabilities, and can debug remote WASP servers as well as the WASP server that is embedded with Eclipse via the plug-in. So long as the remote server has been started in debug mode (a command-line option), then the WASP Management console within Eclipse can attach to that remote server and debug it at runtime.

Building, Deploying, and Testing a WASP Web Service

Now we're ready to walk through the construction, deployment, and testing of a simple WASP Web service. We won't have the opportunity to explore every feature listed above, but we will get a solid sense of WASP Developer's capabilities as a Web service application development platform.

Step 1: Transform a Java Class into a Web Service

1. Open the Web Service perspective (Window –> Open Perspective –> Other –> Web Service.
2. Create a Web Service project (New –> Project –> Select Systinet Web Services on the left, and Web Service Project on the right [see Figure 1]).
3. Click "Next."
4. Supply a project name and click "Finish."
5. Create a simple Java class with an instance field of type String, and a single business

Altova

www.altova.com

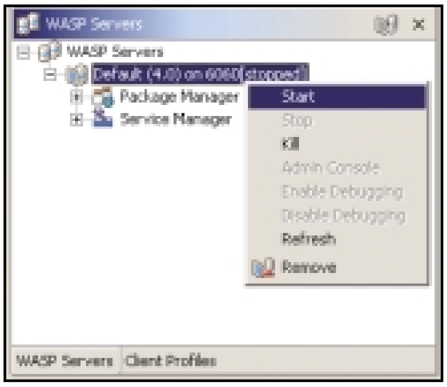


FIGURE 2 WASP Server view

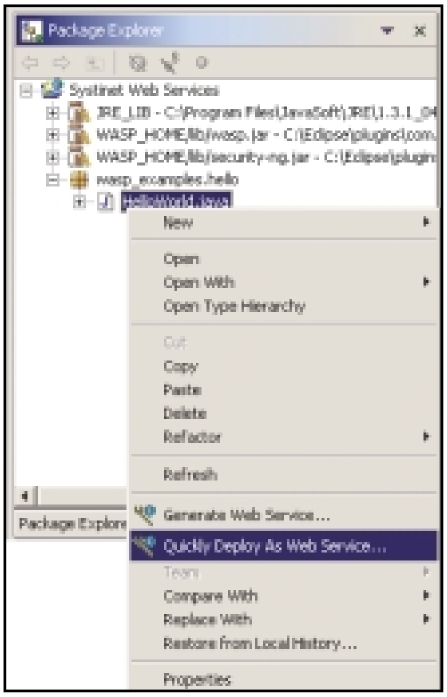


FIGURE 3 Package Explorer view

method, public String getMessage(), that returns the value of the instance field in the body of the method. See Listing 2 if you are unsure of how to do this.

6. Save and compile this class (compilation happens by default in Eclipse when you save). Resolve any errors that appear in the Task view.
7. Repeat the previous step until you get a clean build.
8. Start the WASP Server. In the WASP Server view (bottom left), find the Default WASP server, right-click on it, and select Start (see Figure 2).
9. In the Package Explorer view, right-click on the Java class that you have created, and select "Quickly Deploy as a Web Service" from the context menu (see Figure 3).
10. When the dialog box appears, select the default embedded WASP Server running on port 6060. Click the "Finish" button.
11. Verify the successful deployment of the service by expanding the Package Manager node underneath the Default server in the WASP Servers view. If you can't locate the service, attempt to redeploy it.

Step 2: Generate a Java Client for Your Web Service

1. In the WASP Servers view, expand the Package Manager node underneath the Default server and locate your deployed service. An icon of a wrench and globe indicates the specific service. The text description should begin with '{urn:'. Right-click on this line and select "Create Client" from the context menu (see Figure 4).
2. When the dialog box appears, browse and select the package your class is a member of. Provide a name for the client in the third text field.
3. Click the "Finish" button.
4. A template class will be generated for you, along with local proxy classes that abstract the low-level SOAP messaging details from you.
5. Add a line after the last comment to invoke the getMessage() method on the service object and pass the resulting String object into a System.out.println() command. See Listing 3 for the complete client code.

Step 3: Testing the Client and Service

1. Launch the client by selecting the client class and then clicking the running man icon in the toolbar, or through the menu system: Run –> Run...
2. When the Launch Configurations window comes up, click on the WASP Java Application configuration and click the "New" button.
3. Supply a name for the configuration at the top on the right-hand side.
4. Make sure that the correct project is selected. If not, then click the "browse..." button and select the correct project.
5. Verify that the correct class is displayed in the "Main Class" text field. Locate it through the "Search..." button if necessary.
6. Click the "Run" button to load the launcher configuration and execute the Web service client. In the future, you won't have to go through these steps again, you'll simply "run" the client class and these launcher settings will be used.
7. Watch the console output and verify that the client successfully invoked the service. The string message sent back by the service should be displayed in blue in the console output.

Step 4: Inspecting SOAP with SOAPSpy

1. In the Web Services perspective, look for a tab labeled SOAPSpy in the same win-

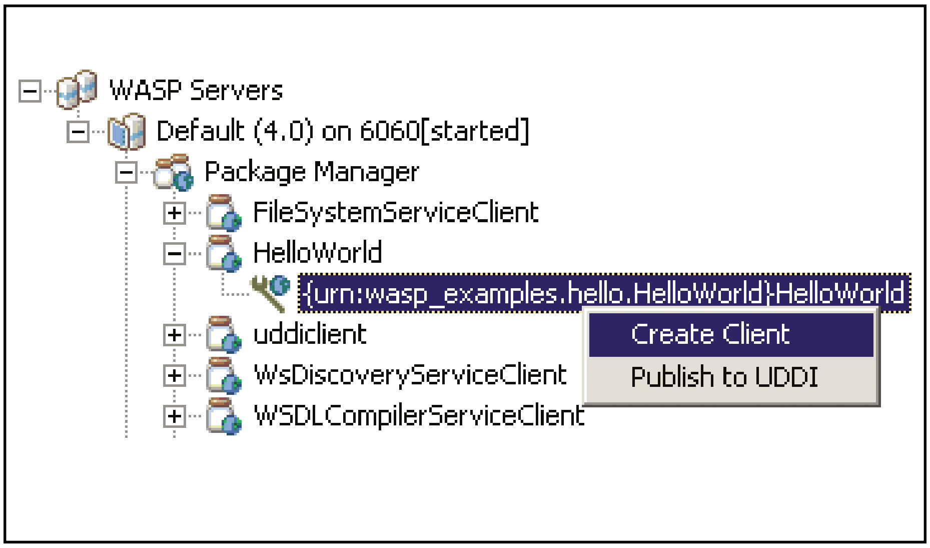
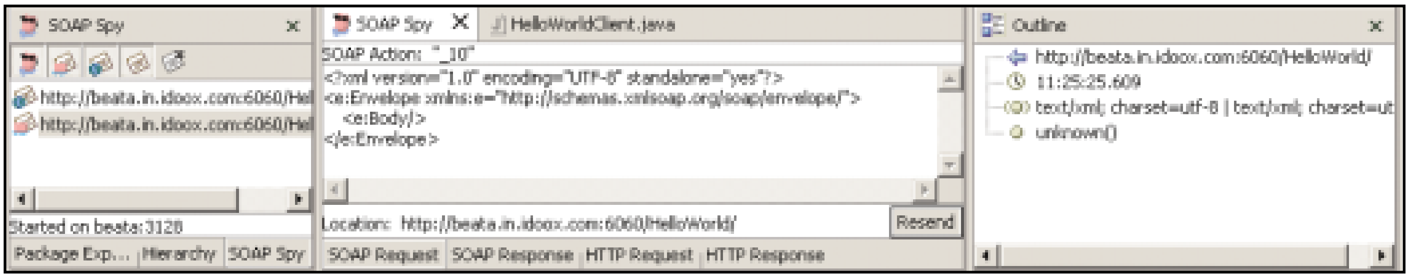


FIGURE 4 Expanded Package Manager node in WASP Server view


SOAP Action: "_10"

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/">
  <e:Body/>
</e:Envelope>
```

Location: http://beata.in.idoox.com:6060/HelloWorld/

http://beata.in.idoox.com:6060/HelloWorld/
11:25:25.609
text/xml; charset=utf-8 | text/xml; charset=ut
unknown()

Started on beata:3128

FIGURE 5 SOAP editor in the editor pane

dow area as the Package Explorer view. Select that tab and the SOAPSpy view will be pulled to the front.

2. According to the documentation, all you need to do to use SOAPSpy is to click the left-most SOAPSpy button at the top of the view (the bar of SOAP with sunglasses). This turns SOAPSpy on. Then you simply run the client application again. However, if you do this, nothing will happen.
3. After digging through several forums on Systinet's Web site, I discovered that the client code needs to be modified so that it uses your computer's actual name rather than localhost. Update both the serviceURI and the wsdlURI in the client source code. Save the changes and rerun the client. This time, you should see one or two entries appear in the SOAPSpy view.
4. By clicking on the entries in the SOAPSpy view, a SOAP editor is opened in the editor pane, allowing you to inspect the SOAP messages exchanged between client and service (see Figure 5). You can even modify the SOAP message and click the "Resend" button that appears at the bottom of the editor. This will circumvent the client and send the SOAP message directly to the targeted service.

Avoiding the Sting of Web Services with WASP

The WASP Server for Java and WASP Developer for Eclipse are mature tools that attempt to take the sting out of developing, deploying, and testing Web service–enabled applications. In this article we've just scraped the surface of what Eclipse and Systinet's WASP Developer can offer Web services developers. With the documentation and the developer resources available on Systinet's Web site (http://dev.systinet.com), you should be well on your way to using these tools to build powerful Web services with relative ease. Enjoy!

Listing 1

```
eclipse -vmargs -
Xbootclasspath/a:ECLIPSE_HOME/plugins/com.systinet.wasp_4.0.
0
/lib/security-ng.jar
```

Listing 2

```
package wasp_examples.hello;


public class HelloWorld {

 protected String message = " Howdy Ya'll ";

 public String getMessage() {
  return message;
 }//end getMessage()
}//end HelloWorld
```

Listing 3

```
package wasp_examples.hello;

import org.idoox.webservice.client.WebServiceLookup;
import org.idoox.wasp.Context;
import wasp_examples.hello.iface.HelloWorld;


public class HelloWorldClient extends Object {

    /** Constructor */
    public HelloWorldClient() {
    }

    /**
    * @param args the command line arguments
    */
    public static void main (String args[]) throws
Exception {
        wasp_examples.hello.iface.HelloWorld service;
   // The actual server name must be used for SOAPSpy to
work
        String serviceURI =
"http://localhost:6060/HelloWorld/";
        String wsdlURI =
"http://localhost:6060/HelloWorld/";

         // init the lookup
        WebServiceLookup lookup = (WebServiceLookup)
Context.getInstance(Context.WEBSERVICE_LOOKUP);
        // get the instance of the Web Service interface
from the lookup
        service = (wasp_examples.hello.iface.HelloWorld)
lookup.lookup(wsdlURI,
wasp_examples.hello.iface.HelloWorld.class, serviceURI);
        // now, call the methods on your Web Service
interface
        //eg. service.getMessage();
     System.out.println( service.echo( "Hi there" ) );
    }
}
```

Download the code at
sys-con.com/webservices

Augmenting EAI with Web Services

Different strengths for different needs

Only a few years ago, concepts in application integration applied to EAI technologies such as messaging oriented middleware (MOM). However, now Web services is the new technology in town. Because Web Services is a different integration paradigm than traditional EAI, opportunities exist for the use of Web services where EAI falls short.

As one Web services writer aptly said, "Both approaches [that is EAI and Web services] are valid but typically not in the same business context. As a result, we do expect that both Web services technology and EAI will complement each other to offer a portfolio of approaches to build composite applications upon. Most likely, this will lead sooner or later to the integration or consolidation of both approaches."

From this article, you will gain the ability first to understand the differences between Web services and traditional EAI technologies, and second, to formulate strategies for augmenting EAI with Web services where it makes sense.

Strengths of Web Services

The primary strength provided by Web services is in establishing integration standards. The benefits of Web services being based on integration standards are flexibility, ubiquity, and convenience of deploying integration solutions. A more detailed look at these integration standards is found in Table 1.

"The promise of Web services is to enable a distributed environment in which any number of applications, or application components, can interoperate seamlessly among and between organizations in a platform-neutral, language-neutral fashion. This interoperation brings heterogeneity to the world of distributed computing once and for all."

Flexibility

Integrating applications using a common protocol to transfer information is not a new concept. Traditional EAI technologies have been doing this since the early 1980's with MOM. However, the transports used with MOM technologies were proprietary and thus tied to a particular middleware vendor. As multiple middleware solutions made their way into an enterprise, it became necessary to build adapters between the various middleware software packages. These bridges made the case for a middleware of middlewares. In a sense, enterprises were in the same boat before the MOMs were installed – the middlewares became the new applications to integrate.

The promise Web services addresses is in providing a cross-platform way to connect various Web services together, even across different Web service technology providers (i.e., Apache, Actional, Microsoft, Sun, etc.).

Ubiquity

With the groundwork laid down by previous Internet communications advancements, Web services paves the way for the pervasive enterprise. The strength here lies in the ability to begin extending applications beyond the boundaries of the current systems infrastructure.

Currently we are in the early stages of accessing applications through information appliances such as cell phones and PDAs. By extending the enterprise out to these devices, new interactions within the enterprise are possible. Now business transactions can take place that were once rel-

TABLE 1: Web services standards

Function	Standard	Status
Messaging	SOAP	Available
Web Service Meta language	WSDL	Available
Repository for lookup and discovery of deployed Web services	UDDI	Available
Security	WS-Security SAML	Under development
Quality of Service	HTTPR	Under development
Transaction handling	WS-Transactions	Under development
Work flow	WSFL WSCI BPEL4WS	Under development

Author Bio
Jim Fisher is a senior partner at the SequenceGroup, an aggressive process and technology consulting firm that focuses on freeing up capital from ongoing operations. He has 12 years of application architecture and development experience and is a recognized expert in the emerging Web services arena. He holds the Web Services Chair at the EAI Industry Consortium, where he guides industry adoption of Web services technology.

Black Hat

www.blackhat.com

egated only inside the corporate firewall.

However, these types of deployments have several barriers to entry. One of these barriers is the existence of numerous architectures and platforms on which these information appliances are run. Implementing and exposing needed services through Web services will facilitate the extension of the integrated enterprise out to these devices.

Once Web services deployments reach a critical mass, the friction encountered in connecting other systems and appliances will be reduced to nil. This will primarily be because companies will see the economies of scale in developing tools to provide this functionality. Also, when Web services integration standards, such as quality of service and security, become established, the enterprise will be extended beyond its traditional boundaries, leading to greater business opportunity and efficiency. The enterprise, which takes advantage of Web services pervasively, will see opportunities that were neither seen nor conceived before .

Convenience

Web services provide a mechanism for easy access and consumption. The convenience rests on the fact that Web services are based once again on standards.

Traditional EAI technologies have progressed to the point where they can expose services by metadata and repositories. However, since the technology is proprietary, the various EAI vendors do not interoperate natively. In a general sense, unless an EAI bridge is utilized, this would mean only TIBCO clients can see and have access to TIBCO integration objects.

With Web services, every application may discover and interoperate with comparatively little or no friction.

UDDI (Universal Description, Discovery, and Integration) provides a standard way for Web services to be deployed and discovered. Third-party tools can integrate with a UDDI repository with little or no compatibility issues. Compared to what it would take to connect one EAI vendor's message broker to another's repository, this is a significant improvement.

A Web service deployment has a standard language for describing its semantics. This language is called WSDL (pronounced by some as "wiz-dul"). The acronym stands for Web Services Description Language. By describing a Web service using WSDL, a potential client of the service can quickly understand how it will communicate with the service. This is particularly handy for Web service tool vendors. Web service development tools can easily use WSDL to describe in detail how clients will need to interact with the service.

Strengths of EAI

The maturity of traditional EAI technologies, on the other hand, provides capabilities that have stood the test of time. Industry demands a reliable, efficient, and robust infrastructure to operate the mission-critical applications within an enterprise.

It should be noted that all of the strengths listed here also come with the caveat that the EAI technologies are proprietary. At best, integration patterns are shared across technologies, but in platform and execution incompatibilities abound. While bridges do occur between EAI technologies, these bridges are point-to-point and not cross functional between other EAI vendors.

Reliability

Traditional EAI technologies typically provide at least two quality-of-service levels for message delivery, a standard delivery and a more dependable delivery service mechanism. Different vendors use different nomenclature, but the concepts are basically the same.

For instance, Vitria labels the standard message delivery as reliable, while for greater dependability in guarding against the case of a server crash, the guaranteed message delivery quality of service is recommended.

Different quality-of-service levels are used depending on the business needs and timeliness of the data in transit. If the relevance of a data message expires within a few seconds, then a high quality-of-service level is probably not needed. A real world example would be the stock ticker example where the real-time nature of the business dictates the necessity of speed. Bringing the server back up after a crash and restoring the stock trades as they came through will probably not be much good. In some trade exchanges, after a few minutes have passed a particular trade price will have no business value.

In contrast, a trade management system, the stock trade journal of record, would not rely as much on the real time aspects, but rather more on the guarantee that entered trades get processed into the trading system. This type of scenario calls for a high quality-of-service level. A stock trader cannot afford to have a few trades get lost somewhere because a thunderstorm causes a power outage.

Efficiency

EAI message transports have the ability to send thousands of message transactions per second on a typical network. This is due largely to the close mapping between the EAI message transports.

When large message throughput is needed, traditional EAI technologies have a comparatively better chance than Web services of being able to efficiently handle the load. Comparatively speaking, the layer of abstraction that Web services provides for integration flexibility leads to greater message data volumes and slower transaction throughput rates. Therefore, efficient message transport is a clear strength for the traditional EAI technology camp.

Some might contend that Web services are efficient because the nature of Web services is both flexible and convenient to develop, leading to lower development costs and thus greater efficiency. While this might be true, there are aspects of project cost other than just development. In fact, studies show that 80% of a system's cost is tied up in the maintenance and support phase of a system's life cycle.

Robustness

As is the case with traditional EAI technologies, mature technologies are often robust because they have stood the test of time. Business issues and problems have challenged the EAI paradigm, causing the EAI vendors to provide solutions to difficult problems. EAI vendors have endured considerable tribulation in getting their software to effectively integrate disparate systems in the face of unforeseen mishaps and catastrophes.

However, through perseverance these EAI

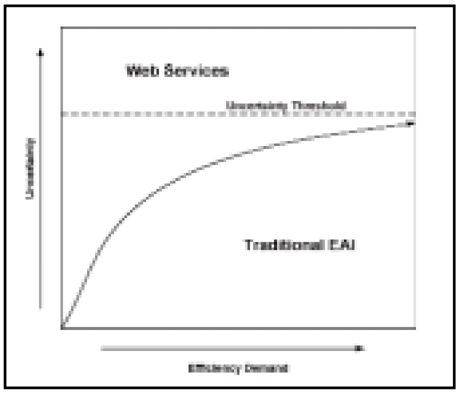


FIGURE 1 **Web services standards**

vendors have developed mature products for accomplishing the enterprise integration business objective.

When to Use Web Services to Augment EAI

Taking Web services in one hand and traditional EAI technologies in the other, system integrators now have access to a greater toolset of technologies in which to provide enterprise solutions. There are times where exposing a system as a Web service is more effective and times when EAI is better.

Given that Web services standards are currently in a state of flux, it is not advisable to integrate the entire enterprise using existing standards. Doing so will likely result in your needing to tear down and rebuild tomorrow. Figure 1 shows an interesting relationship for integration solutions when Uncertainty is plotted as a function of Efficiency Demand.

Uncertainty describes the extent to which a system environment is known and understood. Efficiency points to how well data can be run on a standard network and server configuration. In Figure 1, the less interfaces are understood the more uncertainty one will have in a particular integration scenario. Also, the likelihood that interfaces will change in the future increases the amount of uncertainty. The Uncertainty Threshold depicted in the graph describes an asymptote approached by the line of demarcation separating Web services and EAI. This threshold represents the amount of uncertainty in an integration solution needed to warrant the need of a Web services approach over a traditional EAI approach.

Integrating the Technologies

The remainder of this article documents scenarios where it makes sense to augment an EAI integration strategy with Web services. The scenarios described are:

- Connectivity beyond the corporate enterprise
- COTS software approach to integration
- Synchronous transactions
- High uncertainty compared to efficiency demand

Connectivity Beyond the Corporate Enterprise

In spite of Web services security standards being still under development, connectivity via Web services can be provided to external business partners with relatively secure mechanisms. This scenario is probably the largest and most significant play for Web services in the enterprise. It was not until the development of the World Wide Web that companies began to allow outsiders to access internal systems through the Web browser. In much the same way, Web services will allow convenient access to an organization's enterprise system in a secure and deliberate fashion.

Traditional EAI technologies did provide mechanisms for punching through the corporate firewalls. However, these mechanisms were proprietary, which led to inflexible and inconvenient integration solutions, especially when an organization decided to either change EAI vendors or change a particular line-of-business system providing the Web service to the enterprise.

COTS Software Approach to Integration

Commercial Off The Shelf (COTS) products should take advantage of Web services technologies and begin providing such connectivity in their system via Web services mechanisms. Even as Web services standards change through industry innovation, providing connectivity via Web services would increase the convenience of integrating with the existing enterprise by an order of magnitude.

Synchronous Transactions

Integration scenarios calling for synchronous transactions (i.e., request-reply) are prime candidates for exposure to the enterprise by a Web service. The most common Web service implementations use HTTP as their protocol. Unless a proprietary message provider is employed, the very nature of HTTP requires a demand-driven transaction in which the client must make a call to the service requesting information.

High Uncertainty Compared to Low Efficiency Demand

When high uncertainty exists relative to the efficiency demand of an integration solution, Web services would likely be a better candidate. This scenario plays to Web services' strengths; however, EAI's strengths are less of an advantage in this case.

Since Web services are based on open standards, even changing out the Web services provider should have little impact on the integration with other systems. This type of change could be likened to changing the browser used to view a Web change. Granted there are differences, but most of these differences are minor compared to the differences between the interfaces of line-of-business applications.

Conclusion

From a high level, Web services and EAI seem to offer similar value propositions. As demonstrated here, Web services and EAI provide different strengths to application integration.

The primary benefit of understanding the differences between Web services and traditional EAI solutions is in being able to see where EAI is inadequate and how to provide solutions to these inadequacies with Web services.

In a world where Web services and EAI technologies coexist, opportunities for creative solutions will provide new venues for more effective application integration solutions.

References

- Buyens, Marc. (2002) "Web Services - The discovery of paradise." The *Xpragmatic View*. www.xpragma. com/view49.htm
- Ambler, Scott. (2001) *The Object Primer*, Cambridge University Press.

Beyond SOAP: Optimized Web Services

A more optimal transport with no changes for the client

To most people, the term "Web service" implies a remote server offering some functionality that can be invoked over an HTTP connection, through SOAP-based messages.

AUTHOR BIO:

Howard D'Souza has more than 10 years' experience in software development and is a software architect at Cysive, Inc., where he works on the Cymbio Interaction Server. Howard is a Sun Certified Java Programmer and Developer with a post-graduate diploma in software technology and a bachelor's degree in engineering.

HDSOUZA@CYSIVE.COM.

The ability to penetrate firewalls, as well as not having to install any code on the client side, makes it easy for service providers to quickly deploy and make their services available to a wide audience over a remote network. All that's needed is the service's definition, usually in the form of a WSDL file. While this remains a popular usage pattern for conventional SOAP-based Web services, an area that has been largely ignored is the use of Web services within a company's enterprise. One of SOAP's biggest criticisms has been that SOAP-based Web services don't perform as well as traditional distributed architectures like EJB or CORBA. There is some truth to this. If the client and server are colocated on a fast network, the overhead of pumping messages through SOAP over HTTP may be much more than a traditional remote call.

What is less commonly known is that the Web services standards do not mandate the use of either SOAP or HTTP. In fact, WSDL, which is commonly used to describe and publish Web services, is extensible and allows services to be implemented over virtually any protocol or transport. WSDL provides a language- and protocol-neutral way to describe a service, and allows extensions that can bind this neutral definition to any specific transport. In WSDL terms, the tie between the abstract service definition and its concrete endpoint is called a "binding." It is possible to have multiple bindings for a single service, allowing sophisticated clients running on a fast local network to use the alternate binding to get significantly improved performance. At the same time, the service can offer remote clients the same functionality over SOAP and HTTP. This scenario is depicted in Figure 1.

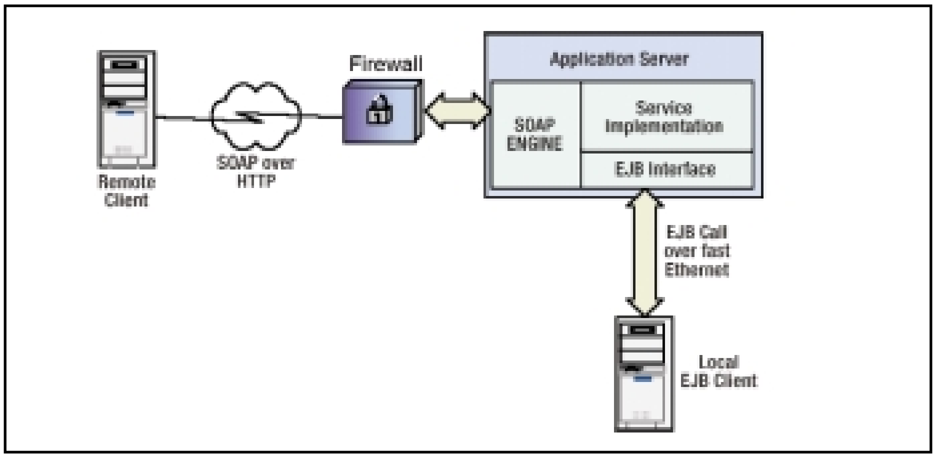


FIGURE 1 | 1 Optimized Web service invocation

Overview of WSDL

To understand how multiple bindings may be defined and exposed for a Web service, let us first explore the Web Service Definition Language. WSDL is a W3C working draft for an XML format for describing network services. A service definition includes an abstract description of the operations and messages that are supported, and their bindings to a concrete protocol.

A WSDL document may define types, mes-

BEA eWorld

www.bea-eworld.com

sages, operations, and port types, which are abstract. It may also define services, ports, and bindings, which are specific to an implementation. Each of these corresponds to an XML element and is described below. Some of these sections are extensible, which allows WSDL to support future protocols and formats.

- ***Types:*** Contains abstract type definitions for the data types used by the operations of the Web service. The most commonly used type system is XSD.
- ***Message:*** Defines an abstract message that has a unique name and a set of parts. Each part refers to an intrinsic type, or a type defined in the types section.
- ***PortType:*** Each portType element has a unique name and defines an abstract set of operations provided by the service. Each operation may define input, output, and fault messages.
- ***Binding:*** Defines concrete message and protocol details for each abstract operation in a portType and can be extended to support new protocols.
- ***Port:*** Each port definition section represents a single endpoint or address for a binding and can be extended to support new protocols.
- ***Service:*** Defines a single service. It groups all the ports for that service together.

JAX-RPC: A Higher-Level Web Service Interface

By providing a means for clients to call Web services through a higher-level interface such as Java, we can bypass the use of SOAP and choose the binding that would be most appropriate. JAX-RPC addresses this need. Since the operations provided by a service are defined in the portType section of the WSDL definition, a client should theoretically be able to invoke operations on the service based on the service's abstract portType definition.

In an effort to distance the user from the details of SOAP messages, a group of leading Java vendors, including Sun, IBM, and BEA, proposed a Java Specification Request (JSR101) that defines a Java API for making XML-based remote procedure calls. This API – called JAX-RPC – is an open standard for exposing and invoking Web services from Java. JAX-RPC enables a service endpoint to be developed using either a Java servlet or an Enterprise JavaBean, and can be extended to other implementations. JAX-RPC provides for mapping services between Java and WSDL. A service provider can run a conversion tool and generate WSDL for a Java interface. This WSDL can be published, and may be used by clients to invoke the service. A client may run a conversion tool on the WSDL for a service, and generate Java interfaces and client stubs for invoking that service. Thus, it is possible for both the service provider and the client to program exclusively to Java interfaces, and use JAX-RPC–compliant conversion tools to provide the plumbing between SOAP and Java. The Apache Foundation's Axis project (http://xml.apache.org/axis) provides a free implementation of WSDL2Java that can be used to generate JAX-RPC–compliant Java interfaces, as well as SOAP stubs for any WSDL document.

For example, consider a WSDL document containing a portType for a stock quote service that had a single operation that took a symbol and returned its price. The WSDL fragment containing the abstract service definition is shown in Listing 1.

If WSDL2Java were run on the service, it would generate a JAX-RPC–compliant Java interface that looked something like:

```
package quote;

public interface StockQuote extends
  java.rmi.Remote
{
    public double getQuote(String symbol)
  throws java.rmi.RemoteException;
}
```

It would also generate a factory interface class for the service, which has methods that create an instance of StockQuote (backed by a SOAP implementation), as well a "locator" that implements this factory as interface (see Listing 2).

A SOAP client stub that implements Stock Quote is also generated. The stub executes the getQuote method by making a SOAP call to the remote server.

A client can use the service without dealing directly with SOAP as follows:

```
    StockQuoteServiceLocator locator = new
     StockQuoteServiceLocator();
    StockQuote quote = locator.getStock
     Quote();
    double price = quote.getQuote("CYSV");
```

The getStockQuote method returns the "default" port, which in this case is SOAP.

Optimized Web Services

It is apparent from the code described previously that the client sees the remote SOAP service as a local Java object. This is great because we can replace the implementation with a more "optimal" protocol without impacting clients. For example, if we knew that the Web service was implemented as an EJB, we could use IIOP or RMI to directly invoke the service, without the overhead of converting to and from SOAP. This would be transparent to the client, but would result in a significant performance improvement.

If the StockQuoteLocator class was enhanced, the getStockQuote method could be changed to return implementations of different endpoints. Consider a stateless session EJB implementation of the above StockQuote service. Its remote interface would look like:

```
package ejbquote;
public interface StockQuote extends
   EJBObject
{
    public double getQuote(String symbol)
     throws RemoteException;
}
```

For simplicity, we define the signature of the getQuote method to be the same as that in the StockQuote interface. To locate the EJB, we must capture its location information in an address-like object. For this purpose, let's define an EJBAddressInfo class.

```
public class EJBAddressInfo
{
    public String jndiURL;
    public String homeName;
    public String homeClass;
    public String remoteClass;
    public Properties jndiProperties;
}
```

Now the StockQuoteLocator object can be modified to add an overloaded version of the getStockQuote method that takes an EJBAdd ressInfo instead of a URL:

```
 public quote.StockQuote
getStockQuote(EJBAddressInfo
portAddress)
throws javax.xml.rpc.ServiceException;
```

A client that wanted to use the EJB binding can be coded as shown in Listing 3.

The locator returns an instance of Stock Quote backed by an EJB client that makes an EJB remote request to fulfill the getQuote operation.

The WSDL Extension Mechanism

This looks great except that the client has to create an EJBAddressInfo object and populate it with details about the EJB. This forces the client to know the specifics of the implementation, and defeats the purpose of using a pure Java interface. It would be nice if the address information could be specified in a way that didn't require the client to know about the service's implementation. That's what the WSDL extensibility mechanism is for. The port section that appears within a service definition in the WSDL document can be extended to describe any kind of address. In the case of EJB, the information that is needed to locate the EJB can be represented in the port section as shown below.

First, let's define an XML namespace for our WSDL extension to the definitions element.

```
<definitions xmlns:ejb="http://schemas.
  myextensions.org/wsdl/ejb" … />
```

Then, add the EJB location information to the port section in the service definition (see Listing 4).

Next, add a binding section for the EJB Remote interface, referencing the StockQuote portType. Since we've kept things simple by assuming the method signatures of JAX-RPC StockQuote and the EJB remote interface are exactly the same, we really don't need any additional binding information. The EJB port's binding looks like:

```
  <wsdl:binding name="StockQuoteEJB
    Binding" type="impl:StockQuote">
    <ejb:binding/>
  </wsdl:binding>
```

Any additional mapping of method names or parameters that is needed can be included in the binding section.

Now we modify our locator implementation to extract all the information it needs from the WSDL. We could publish our WSDL through a UDDI directory or URL, and the client could retrieve and save it locally for better performance. The client code for both the SOAP and the EJB bindings is now the same, and looks like:

```
    File wsdlPath = new File("/services/
     StockQuote.wsdl");
    StockQuoteServiceLocator locator = new
StockQuoteServiceLocator(wsdlPath);
    StockQuote quote = locator.getStock
     Quote();
    double price = quote.getQuote("CYSV");
```

The locator parses the WSDL to extract the endpoint information, and creates the appropriate implementation of the StockQuote service. In the case of EJBs, the implementation retrieves the session bean's remote interface and delegates all calls to it.

Invoking the Optimized Service

For the locator to be able to extract destination information, it needs to parse the WSDL. While a JAXP-compliant XML parser can be used to parse the WSDL, it's easier to use a WSDL library to handle the low-level details–JSR110, which describes a Java API for WSDL parsing. There is an open-source implementation of this JSR called WSDL4J, which is referenced from the Apache Axis Web page.

WSDL4J has built-in extensions for SOAP and provides an extensibility mechanism for defining other types. We can define extensibility elements for handling our EJB extensions, and register them with WSDL4J. When WSDL4J encounters our elements, it will invoke our extensions. Building a WSDL4J extension is beyond the scope of this article, but the source code is well documented.

In the case of EJBs, the StockQuote implementation delegates incoming requests to the EJB's remote interface. This can be achieved with very little code by using Java's Dynamic Proxy feature. All that is required is to implement the InvocationHandler interface in Listing 5. The locator instantiates our dynamic proxy implementation of StockQuote (see Listing 6).

Unlike the generated SOAP stub that is specific to the interface it was generated for, the use of dynamic proxies allows the EJBHandler class to be generic. The same technique could be used for building implementations for any service binding, including local and remote Java objects, JMS or .NET. The client always talks to a generic Java interface and doesn't know or care what the actual service implementation is. The preferred binding for a service can be made configurable, and an implementation could be switched to a more optimized transport, without the client knowing about it.

Conclusion

Using this strategy, a SOAP Web service can be exposed over alternate, more optimal transports without requiring the client code to change. The decision as to which transport is the most optimal is not an easy one. The service locator can be made as sophisticated as needed to decide which binding should be used for a given service. Often, this decision can be based on whether the client and server are on the same network, and an examination of the IP addresses is all that may be needed. An issue that I haven't addressed is the mapping of complex types. In most cases, types that are being passed over SOAP can be mapped directly to JAX-RPC–compliant JavaBean types. Serializers and deserializers may need to be written if the mapping is not straightforward. Most JAX-RPC implementations provide support for user-defined serializers and deserializers as well as generation of Java beans for complex types.

Another issue that must be dealt with is exception handling. Depending on the binding transport that is being invoked, different kinds of exceptions may be thrown. JAX-RPC declares all methods as throwing RemoteException, which wraps the actual exception.

There are also security and transactional implications when multiple bindings are provided. The Web services community is working on resolving and standardizing these issues for Web services in general. Open-source efforts like the WSIF project working toward standardizing multiple Web service bindings, but we have a long way to go before any standards emerge. We are still very early in the life cycle, but optimized Web services looks like something that can greatly enhance the usability of Web services within an enterprise.

References

- *JSR-101 Java API for XML-Based RPC:* http://java.sun.com/xml/downloads/jaxrpc.html
- *JSR-110 Java APIs for WSDL:* http:// jcp.org/en/jsr/detail?id=110
- *Web Services Description Language (WSDL):* www.w3.org/TR/wsdl
- *Apache Axis Project:* http://xml.apache.org/axis
- *Java Dynamic Proxy Classes:* http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html
- *Web Services Invocation Framework (WSIF):* www.alphaworks.ibm.com/tech/ wsif

Listing 1: Abstract service definition

```
<!-- Input and output messages -->
   <wsdl:message name="getQuoteResponse">
      <wsdl:part name="getQuoteReturn" type="xsd:double"/>
   </wsdl:message>
   <wsdl:message name="getQuoteRequest">
      <wsdl:part name="symbol" type="xsd:string"/>
   </wsdl:message>

<!-- portType for the StockQuote service -->
   <wsdl:portType name="StockQuote">
      <wsdl:operation name="getQuote" parameterOrder="symbol">
         <wsdl:input name="getQuoteRequest"
message="impl:getQuoteRequest"/>
         <wsdl:output name="getQuoteResponse"
message="impl:getQuoteResponse"/>
      </wsdl:operation>
   </wsdl:portType>
```

Listing 2: Factory interface class

```
package quote;

public interface StockQuoteService extends
javax.xml.rpc.Service
{
    public java.lang.String getStockQuoteAddress();
    public quote.StockQuote getStockQuote()
throws javax.xml.rpc.ServiceException;
    public quote.StockQuote getStockQuote(java.net.URL
portAddress)
throws javax.xml.rpc.ServiceException;
}
```

Listing 3: Overloaded version of getStockQuote method

```
    EJBAddressInfo ejbInfo = new EJBAddressInfo();

    // set the values of the EJB location in ejbInfo …

    StockQuoteServiceLocator locator = new
StockQuoteServiceLocator(ejbInfo);
    StockQuote quote = locator.getStockQuote();
    double price = quote.getQuote("CYSV");
```

Listing 4: EJB location informationadded to port section

```
  <wsdl:service name="StockQuoteService">
<!-- SOAP Port -->
    <wsdl:port name="StockQuoteSOAPPort"
binding="impl:StockQuoteSoapBinding">
         <wsdlsoap:address
location="http://localhost:8080/axis/services/StockQuote"/>
    </wsdl:port>

<!-- EJB Port -->
    <wsdl:port name="StockQuoteEJBPort"
binding="impl:StockQuoteEJBBinding">
     <ejb:port>
      <address location="jnp://localhost:1099">
        <property name="jndi-name">ejb/MyStockQuote></proper-
ty>
        <property name="initial-ctx-factory">
            org.jnp.interfaces.NamingContextFactory</property>
        <property name="home-interface">
ejbquote.StockQuoteHome</property>
        <property name="remote-
interface">ejbquote.StockQuote</property>
      </address>
     </ejb:port>
    </wsdl:port>
  </wsdl:service>
```

Listing 5: InvocationHandler interface

```
public class EJBHandler implements
java.lang.reflect.InvocationHandler
{
    // …
    private File wsdlFile = null;
    private Object ejbStub = null;

    public EJBHandler(File wsdlFile)
    {
        /*
         Parse the WSDL file, extract the EJB destination
         info.
         Retrieve the EJB’s home interface. From it, retrieve
         the remote interface, and assign it to ejbStub.
         Be sure to use PortableRemoteObject.narrow() to get
         the correct home and remote interface types …
        */
    }

    // Called when any method is invoked on the target inter-
face
    public Object invoke(Object proxy, Method method, Object[]
args)
        throws Throwable
    {
        Method instanceMethod =
ejbStub.getClass().getMethod(method.getName(),
            method.getParameterTypes());
        return instanceMethod.invoke(ejbStub, args);
    }
}
```

Listing 6: Dynamic proxy implementation of StockQuote

```
public class StockQuoteServiceLocator
{
  File wsdlFile; // Initialized in the constructor
  //…
    public StockQuote getStockQuote()
    {
        // If the preferred port is EJB Instantiate
        // an EJBHandler proxy and pass it the WSDL file
        return java.lang.reflect.Proxy.newProxyInstance(
            StockQuote.class.getClassLoader(),
            new Class[]{quote.StockQuote.class},
            new EJBHandler(wsdlFile));
    }
}
```

Download the code at
sys-con.com/webservices

introductory
subscription offer!

.NET

A TRULY INDEPENDENT VOICE IN THE WORLD OF .NET

.NET Developer's Journal is the leading independent monthly publication targeted at .NET developers, particularly advanced developers. It brings .NET developers everything they need to know in order to create great software.

Published monthly, ***.NET Developer's Journal*** covers everything of interest to developers working with Microsoft .NET technologies – all from a completely independent and nonbiased perspective. Articles are carefully selected for their prime technical content – technical details aren't watered down with lots of needless opinion and commentary. Apart from the technical content, expert analysts and software industry commentators keep developers and their managers abreast of the business forces influencing .NET's rapid development.

Wholly independent of both Microsoft Corporation and the other main players now shaping the course of .NET and Web services, ***.NET Developer's Journal*** represents a **constant, neutral, expert voice** on the state of .NET today – the good, the bad, and the ugly...no exceptions.

SYS-CON MEDIA

SUBSCRIBE ONLINE!
www.sys-con.com/dotnet/
or Call
1 888 303-5282

Here's what you'll find in every issue of .netdj:

- Security Watch
- Mobile .NET
- .NET Trends
- Tech Tips
- Standards Watch
- Business Alerts
- .NET News
- Book and Software Announcements

.NET Developer's Journal is for .NET developers of all levels, especially those "in the trenches" creating .NET code on a daily basis:

- For beginners:
  Each issue contains step-by-step tutorials.
- For intermediate developers:
  There are more advanced articles.
- For advanced .NET developers:
  In-depth technical articles and columns written by acknowledged .NET experts.

Regardless of their experience level, *.NET Developer's Journal* assumes that everyone reading it shares a common desire to understand as much about .NET – and the business forces shaping it – as possible. Our aim is to help bring our reader-developers closer and closer to that goal with each and every new issue!

SAVE 16%
OFF
THE ANNUAL COVER PRICE

Get 12 issues of .NETDJ for only $^{\$}69^{99}$!

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

ANNUAL COVER PRICE:
~~**$83.88**~~
YOU PAY
$^{\$}69^{99}$
YOU SAVE
$13.89
OFF THE ANNUAL COVER PRICE

LinuxWorld
Conference & Expo

www.linuxworldexpo.com

LinuxWorld

Conference & Expo

www.linuxworldexpo.com

Reviewed by Joseph A. Mitchko

J2EE Release 1.4

Industry standard now provides Web Service Functionality

About the Author:
Joe Mitchko is a technology specialist working for a leading Internet service company and is product review editor and contributing writer for Web Services Journal magazine.
JMITCHKO@RCN.COM

The Web service-based functionality provided in Sun Microsystems J2EE 1.4 is a culmination of the Java APIs and utilities that were previously part of the Java Web Services Developer Pack (WSDP) and have been repackaged for the most part into J2EE 1.4. For the sake of this review, I will cover only the new Web services–related features included in the new release. With that said, let's explore what J2EE 1.4 has to offer regarding developing and implementing Web services.

First, there are plenty of Java-based development platforms out there that enable you to expose J2EE applications as Web services, so the technology is not really all that new. For instance, the concept of using servlets to consume and process Web service requests and relying on EJB components to handle the business logic and persistent data storage is proven technology. What is exciting is that this technology is now officially part of the J2EE standard, and will enable interoperability of Web service functionality across various vendor platforms. What this promises for developers is that instead of using proprietary APIs that come with a particular Java application server, you will be using standard J2EE APIs. So, just as JSPs and EJBs are standardized, you will be seeing the same for Web service–specific APIs.

In addition to adding Web services functionality, J2EE 1.4 comes packaged with some of the more popular Java-based open-source software products, including Jakarta Ant and the Cloudscape relational database. Install and configure Apache Tomcat, which also comes with the release, and you have the makings of a basic, roll your own, Web service development platform. You get some really nice features, and you can't beat the price. You need to remember though that large-scale production environments will still require the use of industry-proven application servers such as WebLogic or WebSphere. What's important here is the standardization or J2EE branding of the Web service Java APIs.

COMPANY INFO
Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
Phone:1-800-555-9SUN
or 1-650-960-1300
Web: http://java.sun.com
E-mail: contact-us@sun.com

EVALUATION DOWNLOAD
http://java.sun.com/j2ee/download.html

LICENSING INFORMATION
http://servlet.java.sun.com/help/legal_and_licensing

TESTING ENVIRONMENT
OS: Windows 2000 Professional
Hardware: IBM ThinkPad T30

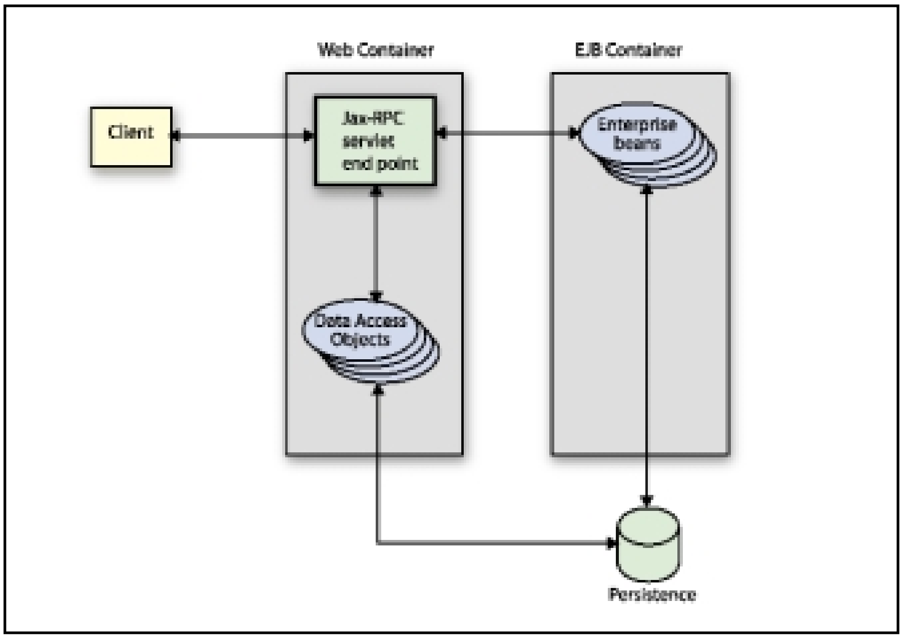


FIGURE 1 Synchronous services with JAX-RPC

The APIs

OK, at this point you may be asking, what are these APIs? Let's go over some of them. First, what I call the "JAX in a box" APIs is JAXP. To those familiar with Java-based XML parsing (SAX and DOM) and XSLT processing facilities, these are really our old friends Xerces and Xalan (or is that

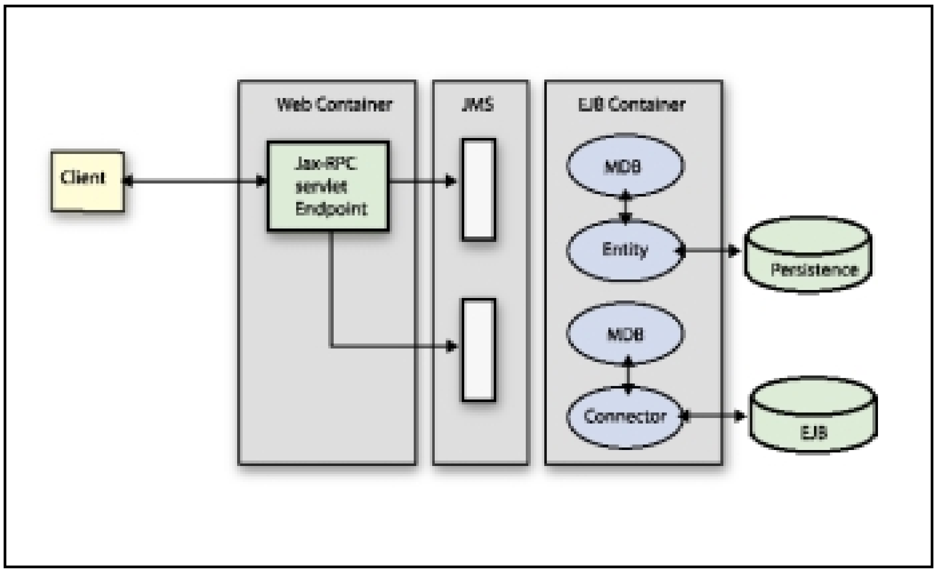


FIGURE 2 **Asynchronous Web service communication**

Xalan and Xerces, I can never get them straight). Both of these started in the open-source arena and now join the J2EE family, which they rightfully should. It is difficult to name any Java-based XML or Web services product in the market that doesn't include some aspect of their functionality and libraries.

Next on the list to join J2EE is JAXR. This API provides you with the ability to look up Web services and other XML-based documents on ebXML and UDDI-based XML registries. JAXR also provides rich query capabilities as well as rich metadata capabilities for classification and association. It gives programmers the capability to write abstract registry client programs that are portable across different target registries

Next in the JAX lineup is JAX-RPC. This API covers all aspects of implementing RPC-type SOAP messages, and for the J2EE 1.4 release includes support for SSL-based authentication, serializers and deserializers for collection types, and other improvements. You would use the JAX-APIs when developing your Web service endpoints. It's basically the servlet-based stuff that maps SOAP parameter types from XML to Java data types, and manages the actually method calls that implement the service.

In a break from the JAX-based naming convention, we will next cover SAAJ, or SOAP with Attachments API for Java. Here we're talking about a number of APIs to cover document-based SOAP service calls that involve the processing of a SOAP message through asynchronous message queues. The technologies include DOM4J, Jakarta Common Logging, JAXP, JavaMail, and JavaBeans Activation Framework.

In addition to the APIs, J2EE 1.4 includes a number of utilities for developing and deploying Web services. These include J2EEC, which helps you to generate stubs, ties, and so on for JAX-RPC–based services and WSCOMPILE, which also helps you to generate stubs, ties, and WSDL files for JAX-RPC services.

Using J2EE 1.4

Now that we have gone over what the J2EE 1.4 release contains, how would you use it? Here's an example of how all the parts come together using the APIs provided in J2EE 1.4.

Figure 1 shows a synchronous Web service using the JAX-RPC APIs. The Web service endpoint is managed by a JAX-RPC–based servlet that accesses persistent storage (a database) directly or, in the case of calling an EJB component, indirectly. In the case where you are building off of an existing J2EE application, the SOAP endpoint is considered by the system to be just another view or presentation of the application.

Figure 2 shows another example of using the J2EE Web service APIs, this time implementing an asynchronous, message-based service. Clients of asynchronous Web services do not need to wait around for a response from the service. Instead, the response will come some time later in the form of a direct SOAP call to the client (the client has a service also), or by the client polling on a regular basis for the response to arrive. In this diagram, you will notice that the application is using JMS (Java Messaging Service) to deliver the SOAP XML document tied to message-driven enterprise JavaBeans (MDB).

There are a number of excellent white papers available at the java.sun.com site to help you learn how to the use these APIs and how to implement various Web services using the J2EE architecture. "Using Web Services Effectively" not only describes the purpose for each Web service–based API described here, but also provides information on using J2EE patterns and other informative tips, such as how to directly expose an EJB component as a Web service endpoint.

In addition to documentation, you can download a sample application involving a Pet Store order management system. This provides an excellent start for anyone interested in developing Web services using the framework.

The J2EE Release 1.4 installation file and associated documentation can be downloaded from the Web site at http://java.sun.com/j2ee/download.html. While you're at it, you'll also need to download and install the 1.4 version of the Java 2 SDK before installing J2EE. You will need to become a registered member of the Java Developer Connection, which is not that difficult (you can sign up and get started right away).

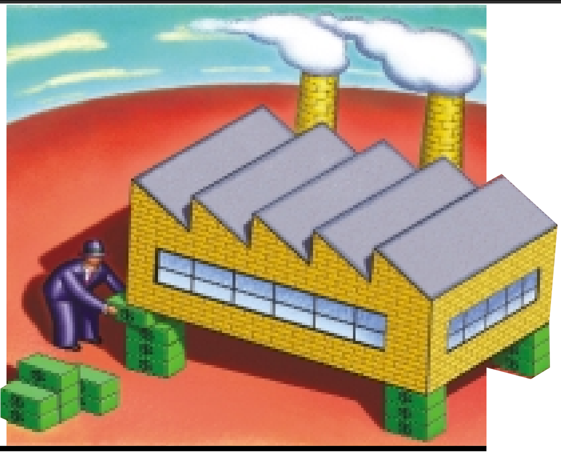
At the time of this writing, I was looking at a beta version that was open to everyone in the Java developer community.

Conclusion

Compared to other Java-based Web service development products on the market today, developing SOAP services with J2EE 1.4 requires a lot more hands-on work, and heaven forbid, you may need to actually read the XML. But for those projects that require highly customized SOAP-based Web services, especially on top of existing J2EE frameworks, the ability to work at the API level can't be beat. J2EE 1.4 also provides an excellent start for those interested in understanding Web service and XML processing basics.

Putting Web Services into (Business) Context

Avoid excess technology while adding value to the business



Web services tool vendors frequently compete on how quickly their users can "generate a Web service from scratch" or "expose a Java/COM+/CORBA class as a Web service." While speed of development is important, the broader business needs of an enterprise must be the main driver of new technology adoption.

AUTHOR BIO:

Brent Carlson is vice president of technology and cofounder of LogicLibrary. He is a 17-year veteran of IBM, where he served as lead architect for the WebSphere Business Components project and held leadership roles on the "IBM SanFrancisco Project." Brent is the coauthor of San Francisco Design Patterns: Blueprints for Business Software *and* Framework Process Patterns: Lessons Learned Developing Application Frameworks.

Blindly applying new technology ultimately results in more poorly conceived software. These applications are just as difficult to maintain as existing applications, only they're running on yet another technical infrastructure. As ZapThink, LLC, an XML- and Web services–focused industry analyst group, explains, "Just because a new technology has promise doesn't guarantee that it will be applied correctly."

So how do IT organizations move from "playing" with Web services development to providing truly useful services that support key business objectives? Companies must be able to view their existing software development assets (SDAs) within the context of the enterprise's strategic business processes in order to take full advantage of the promise of Web services. IT executives need to understand what assets exist, where they are located, and how each fits into the corporate business landscape, i.e. a model-based approach. This approach to Web services development gives enterprises an efficient way to enable existing SDAs as Web services; it can be broken into four steps – assess, build, locate, and employ.

Let's take a simple example – a currency conversion component – through this process.

Example Component – CurrencyExchange EJB

Our example component is an EJB that provides currency conversion services – converting dollars to euros, setting up conversion tables, and so on. Figure 1 presents this example component in UML, showing the external interfaces of the component without any of the implementation details involved in building the component.

The functional capabilities of this component are presented to its users via a SessionBean interface containing a number of method definitions. These methods place dependencies on two supporting elements of the external component definition: a set of transfer object types used

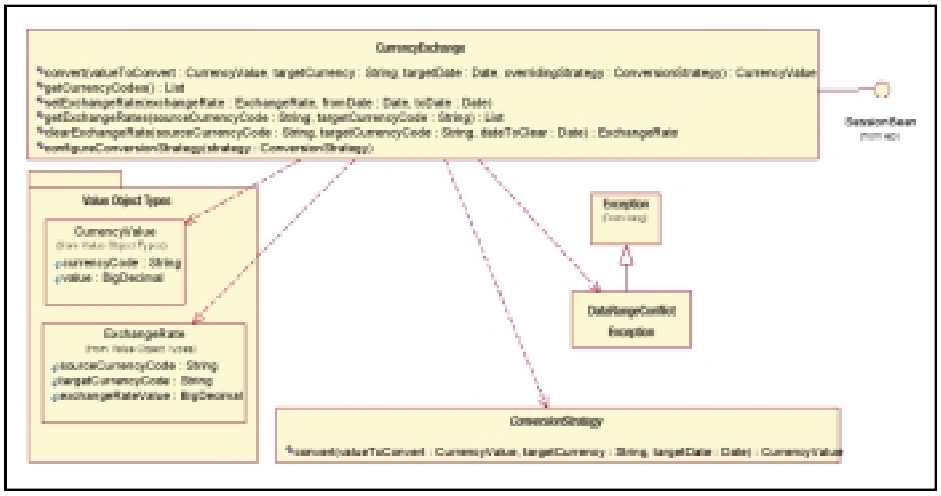
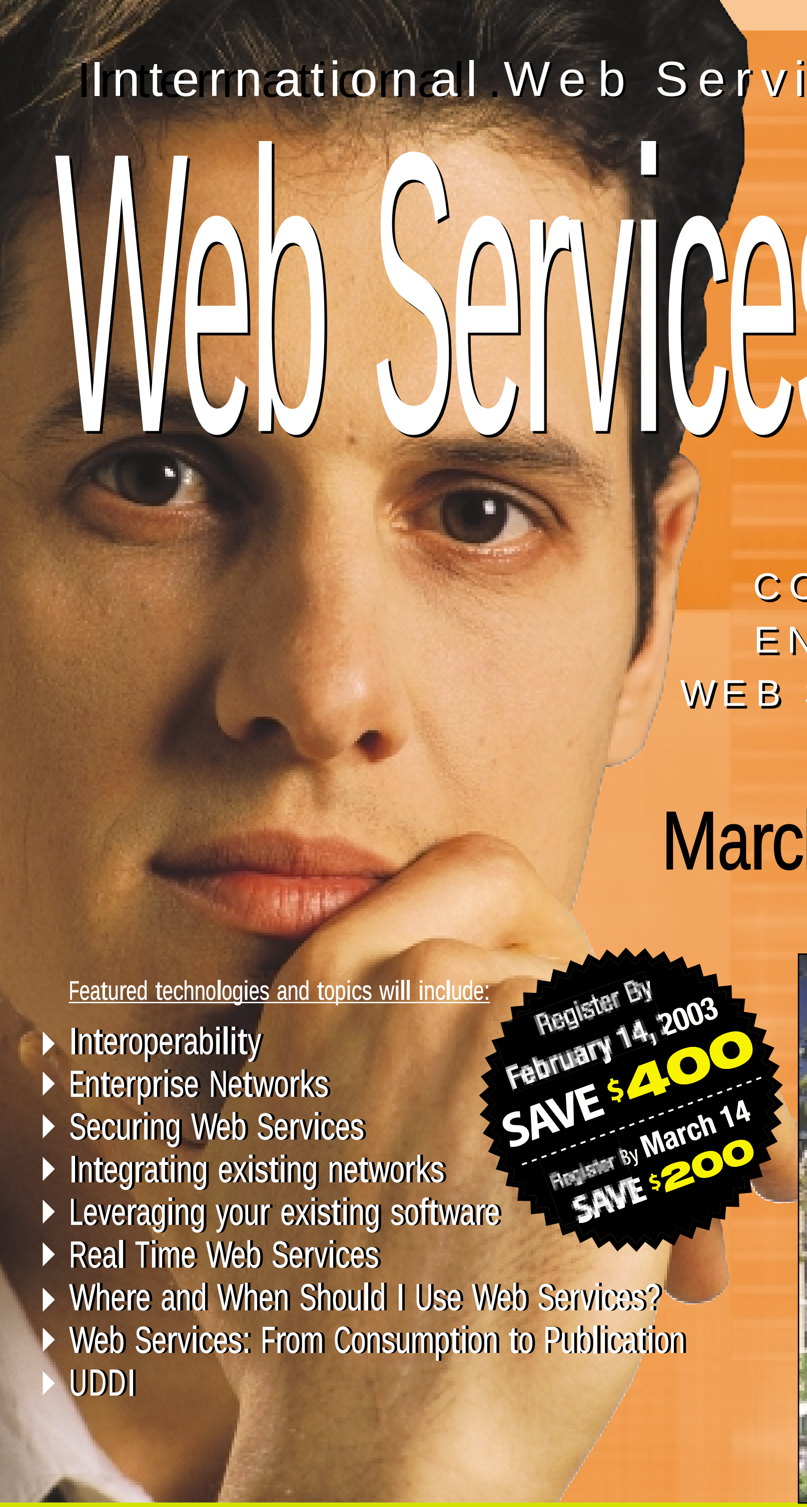

CurrencyExchange
convert(valueToConvert : CurrencyValue, targetCurrency : String, targetDate : Date, overridingStrategy : ConversionStrategy) : CurrencyValue
getCurrencyCodes() : List
setExchangeRate(exchangeRate : ExchangeRate, fromDate : Date, toDate : Date)
getExchangeRates(sourceCurrencyCode : String, targetCurrencyCode : String) : List
clearExchangeRate(sourceCurrencyCode : String, targetCurrencyCode : String, dateToClear : Date) : ExchangeRate
configureConversionStrategy(strategy : ConversionStrategy)
SessionBean
Value Object Types
CurrencyValue
(from Value Object Types)
currencyCode : String
value : BigDecimal
ExchangeRate
(from Value Object Types)
sourceCurrencyCode : String
targetCurrencyCode : String
exchangeRateValue : BigDecimal
Exception
(from lang)
DateRangeConflict Exception
ConversionStrategy
convert(valueToConvert : CurrencyValue, targetCurrency : String, targetDate : Date) : CurrencyValue

FIGURE 1 CurrencyExchange component



International Web Services Conference & Expo

Web Services Edge 2003

web services EDGE conference&expo

JDJ EDGE conference&expo

XML EDGE conference&expo

.NET EDGE conference&expo

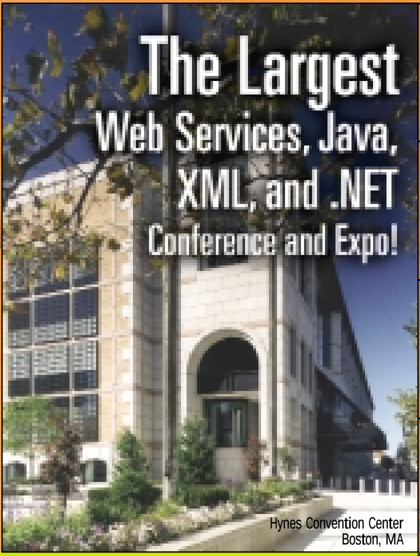
CONNECTING THE ENTERPRISE WITH WEB SERVICES, JAVA, XML, AND .NET

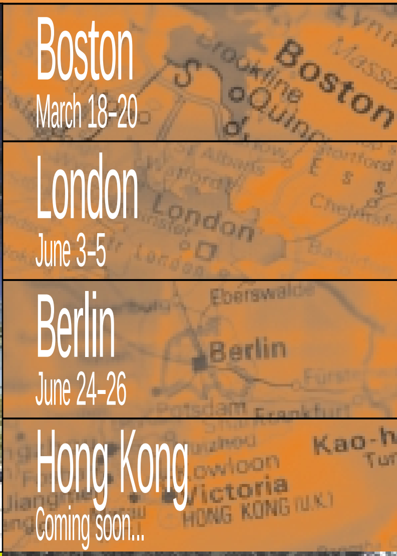
March 18-20, 2003
Boston, MA

Featured technologies and topics will include:

- Interoperability
- Enterprise Networks
- Securing Web Services
- Integrating existing networks
- Leveraging your existing software
- Real Time Web Services
- Where and When Should I Use Web Services?
- Web Services: From Consumption to Publication
- UDDI

Register By February 14, 2003 SAVE $400
Register By March 14 SAVE $200



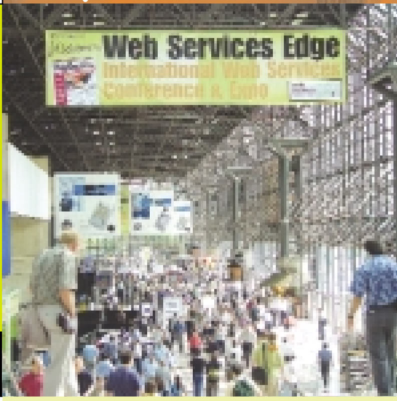


For more information visit
www.sys-con.com

Over 200 participating companies will display and demonstrate over 500 developer products and solutions.

Over 3,000 Systems Integrators, System Architects, Developers and Project Managers will attend the conference expo.

Over 100 of the latest sessions on training, certifications, seminars, case-studies, and panel discussions promise to deliver REAL Web Services Benefits, the industry pulse and proven strategies.



Contact information: U.S. Events: 201 802-3069 or e-mail grisha@sys-con.com • European & Asian Events: 011 44 208 232 1600

OWNED BY SYS-CON MEDIA
PRODUCED BY SYS-CON EVENTS

Conference program available online!
www.sys-con.com/webservices2003east

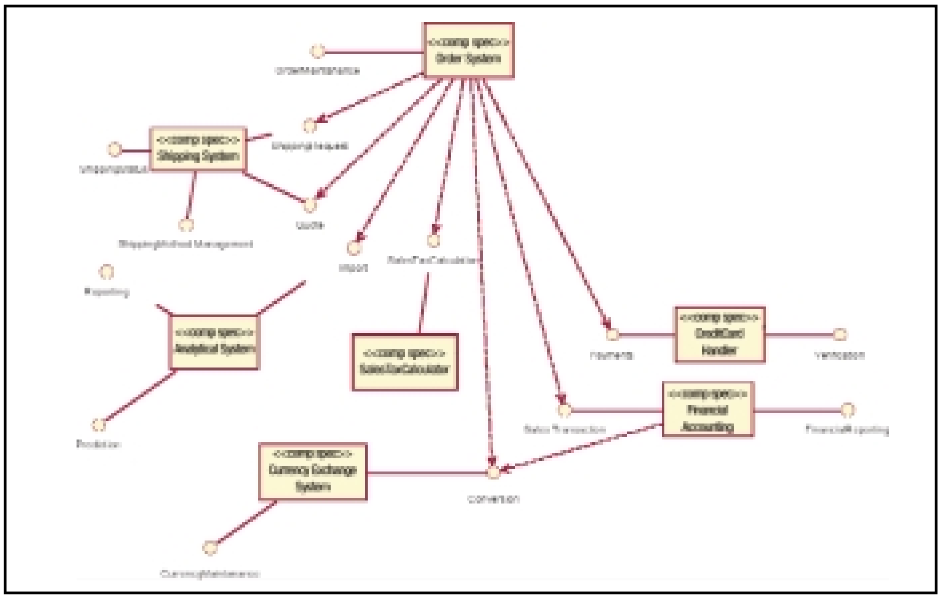


FIGURE 2 W-commerce reference model fragment

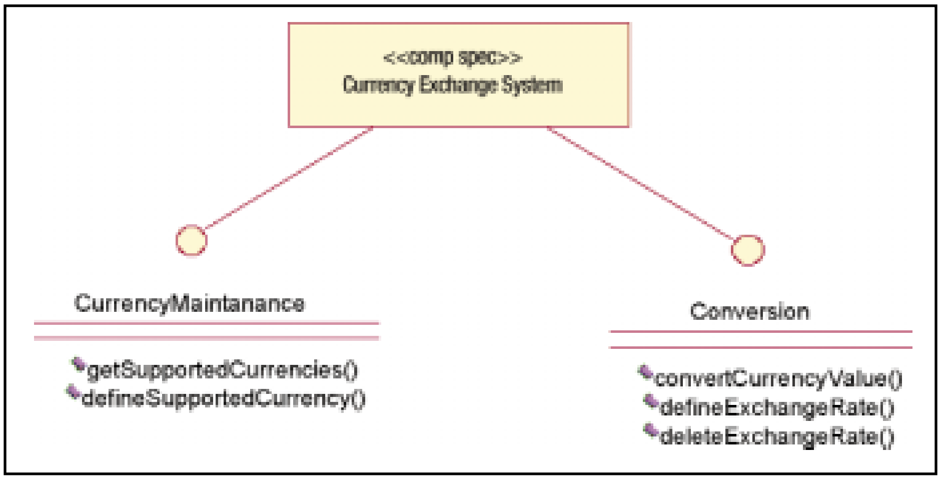


FIGURE 3 Currency Exchange System reference component detail

to interchange nonprimitive data with the component, and a specialized exception that users of the component must handle when setting exchange rates into the component. Also, our CurrencyExchange component allows its users to control the conversion algorithm or algorithms used when converting one currency to another, both through a component configuration interface as well as by allowing the configured conversion strategy to be selectively overridden on a call-by-call basis.

Applying the enABLE Methodology

The four steps in our model-based approach are:

- ***Assess*** what you have and create a business roadmap for migrating to Web services
- ***Build*** a catalog of essential software development assets (SDAs) mapped to your business roadmap (i.e., your business architectures and models)
- ***Locate*** the most appropriate software assets for your high-priority services using the catalog you have built
- ***Employ*** these assets in your tools of choice for developing Web services

Assess

A typical IT organization will have thousands of assets that have accumulated over the years. Assessing which of these assets are of value to the organization moving forward must incorporate both business and technical perspectives. Issues that need to be taken into account during the assessment phase include:

- The technology used to build the asset
- Compatibility of the asset's technology with future technical architectures being defined by the organization
- Level of documentation (i.e., artifacts resulting from the software development process) available that describes the asset's business functionality
- Current use of the asset to support the organization's business activities
- Expected use of the asset to support future business needs

As you begin, remember the 80/20 rule; some SDAs are obvious candidates for your initial cataloging efforts, while other marginal assets are best left behind.

How does our example component fit against our assessment criteria?

- ***Technology:*** Built using J2EE technology.
- ***Future technical architectures:*** Our organization is in the process of defining a service-oriented architecture that includes Web services as a core element. These Web services will be implemented using a combination of J2EE component technology and adapter technology, which encapsulates legacy applications within our environment.
- ***Level of documentation:*** Component artifacts include source code, a deployable .jar file, a UML model of the component that includes both an external client-oriented view as well as an internal design-oriented view, and javadoc.
- ***Current use of asset:*** Component was built as part of a project to incorporate international capabilities into our order management application.
- ***Expected use of asset:*** As our business expands its international capabilities, the need for a currency conversion service available to multiple departments has become clear.

Based on our assessment, it appears that our currency conversion component is an important asset that should be preserved and managed into the future. This leads us to the next stage of our model-based approach.

Build

In the build phase, our task is to align our

TABLE 2: Operations definitions

Reference Component operation	EJB method
getSupportedCurrencies	getCurrencyCodes
defineSupportedCurrency	No direct equivalent
convertCurrencyValue	convert
defineExchangeRate	setExchangeRate
deleteExchangeRate	clearExchangeRate

essential SDAs with our business roadmap (i.e., our business architectures and models describing current and future business requirements to be placed on the IT organization). The end result of this effort is a ready reference that gives us the ability to see what aspects of our business architecture are already supported, to identify redundancies, and to see where gaps exist. Ultimately, effective use of this reference model will enable us to build Web services that present the right level of information and interact with the right underlying business systems.

Our business architecture team has been busy defining requirements for our next generation of e-commerce systems, and they have taken the next step to express these requirements in the form of a UML reference model. Figure 2 illustrates a portion of that model that is pertinent to our example component.

Each class within this UML diagram represents a coarse-grained reference component that supports one or more interfaces. If we look into the Currency Exchange System component in more detail, we see that its two interfaces define a series of operations as shown in Figure 3.

The operations defined here can be mapped against the methods defined by our EJB (see Table 1).

Notice that some of our reference component methods are not directly supported by our EJB, and also that some of the capabilities of our EJB (e.g., our ability to configure conversion strategies) aren't reflected in our reference model. However, our mapping does show that we have a strong affinity between our desired business capabilities and this existing SDA.

Locate

As our development team is tasked to build out portions of our next-generation business architecture, they will take advantage of our previously built business reference model to search for candidate SDAs that can help them do their work more efficiently. Let's assume that our team has the responsibility to build out a Web service supporting currency conversion. Their work activities flow as follows:

1. They begin their work by identifying the portion of the business reference model that applies to their project – specifically, the Currency Exchange System.
2. They then investigate the services and operations defined by that reference component.
3. For the set of selected operations, they initiate a search against the underlying SDAs mapped against the reference model. In this case, they discover our CurrencyExchange EJB.
4. The team then retrieves various artifacts as part of their evaluation process, ultimately concluding that they can take advantage of this asset in their Web services development project.

How will they investigate the components and services associated with this model? While some groups have built reference models to asset mappings through spreadsheets and other similar tools, an SDA mapping and discovery engine can speed and improve this process dramatically.

Depending upon the area of the model being investigated, our team might find multiple assets that support a set of business capabilities, at which point they need to determine if they should select one of these assets for future development or if they need to provide an encapsulation service that binds all of the existing assets together and ensures consistent data and behavior across them (if, for example, multiple customer information systems must be supported because of business merger activity in the past). Or they might discover that no existing assets support a portion of our model, indicating that "green-field" development is needed to support some expanded business capabilities. Regardless of the outcome, our team has a better view of what is available to them to use in the next stage of the process.

Employ

The employ stage is where all of today's "whiz-bang" Web services tooling comes into play. With our Currency Exchange example, we will likely retrieve our EJB .jar file from its repository location and deposit it into a project within our IDE of choice, then use that IDE's Web service–generation capabilities to convert our SessionBean interface into a Web service with an associated WSDL file, Java client proxy, and other artifacts.

As part of the employ phase, we have some decisions to make:

- How should we expose the information from our specialized date range exception to our Web service client?
- Do we choose to expose our conversion strategy capability on the Web service?
- Do we create separate Web services for our currency conversion operations from our currency maintenance operations, or do we combine them into one service?

Ultimately, as these newly built Web services are created and deployed, our team will complete the life cycle by feeding them back into their catalog, mapped against the business roadmap for which they were built. These Web services now become part of the developer's everyday toolkit – resulting in faster, higher quality, less expensive application development and integration.

Summary

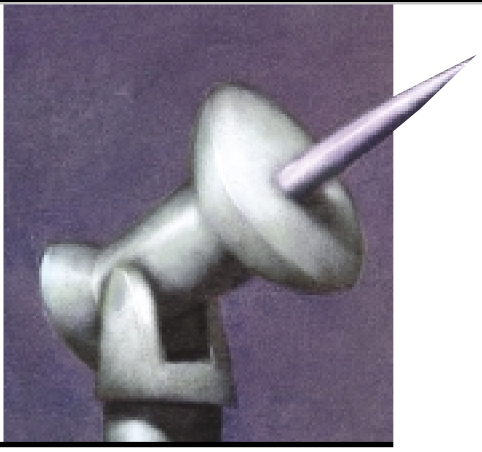
Taking the time to put our Web services and the underlying SDAs that already exist in our infrastructure into an IT library will ensure that the services we produce will provide true value to the business and not introduce more technology for technology's sake.

References

- Schmelzer, R., and Bloomberg, J. ZapFlash, "Understanding the Real Costs of Integration": www.zapthink.um/flashes/10232002flash.html
- Alur, Crupi, and Malks. (2001) *Core J2EE Patterns; Best Practices and Design Strategies*, p. 261. (Prentice Hall PTR).
- Gamma, Helm, Johnson, and Vlissides. (1994) *Design Patterns: Elements of Object-Oriented Software*, p. 315. Addison-Wesley.

Using JAX-RPC Parameter Modes With Apache Axis

A look at both the simple and the complex



The RPC mechanism enables a remote procedure call from a client to be communicated to a remote server. SOAP RPC supports three types of parameter modes – IN, OUT, and INOUT – for remote method invocation. In this article I'll explain how JAX-RPC supports SOAP RPC parameter modes and how to use JAX-RPC parameter modes with Apache Axis.

AUTHOR BIO:

Aravilli Srinivasa Rao, a software analyst at Hewlett-Packard, is technical lead for the development of HP's public UDDI Registry. He is currently involved in a feasibility study of the projects in the mobile space to implement Web services. Aravilli holds a master's degree in computer applications.

SRINIVASA.RAO.ARAVILLI@HP.COM

JAX-RPC

JAX-RPC provides a means for performing RPC over SOAP. It provides rules for client code generation, SOAP Binding, mapping from Java to WSDL and WSDL to Java, type mappings between Java and XML data types, and client APIs for constructing Web service invocations.

JAX-RPC defines one static model and two dynamic models for the client side to invoke a remote procedure. In the static model, stubs are used and the code-generation tools generate the stubs. One dynamic model is based on generating a proxy object dynamically using java reflection APIs and another is based on Dynamic Invocation an Interface (DII) using the Call object. The examples shown in this article use DII rather than the static model.

JAX-RPC interfaces

On the client side, JAX-RPC defines the following major interfaces:

- ***ServiceFactory:*** Creates the instances of Service interface.
- ***Service:*** Defines methods for dynamic proxy invocation and DII. getPort( ) method is used for dynamic proxy invocation.
- ***Call:*** createCall( ) method on Service interface returns the Call object which is used for DII.

JAX-RPC supports the following three types of invocation models:

- ***Synchronous request-response mode:*** A Client invokes a remote method on a service and receives return values or an exception. Client blocks the thread until it gets a response from the remote service.
- ***One-way RPC Mode:*** A Client invokes a remote method on a service endpoint in the one-way mode. The client invocation thread does not block and continues execution without waiting for this remote method invocation to be processed by the service.
- ***Non-blocking RPC invocation:*** A service client invokes a remote method on a service endpoint and continues processing in the same thread without waiting for the return of the remote method invocation.

The Synchronous Request-Response Mode and One-Way RPC Mode are achieved by the JAX-RPC DII Call interface.

JAX-RPC Parameter Modes

JAX-RPC uses *pass by copy* semantics for parameter passing in a remote method invocation. The JAX-RPC specification does not support the *object-by-reference* mode for remote method invocations. JAX-RPC specifies the following rules for the IN, OUT and INOUT parameter passing modes and return value:

- An **IN** parameter is passed as copy. The value of the **IN** parameter is copied before a remote method invocation.
- The **return** value is created as a copy and returned to the caller from a remote method invocation.
- The **OUT** and **INOUT** parameters are passed by copy. JAX-RPC uses the Holder classes to support the **INOUT** and **OUT** parameters. A service client provides an instance of a Holder class that is passed by value for either out or inout parameter. The contents of the Holder class are modified in the remote method invocation and the service client uses the changed contents after the method invocation returns.

- JAX-RPC defines **ParameterMode** class with static instances namely IN, OUT, and INOUT, to specify parameter-passing modes in Web service invocation. **ParameterMode** class acts as a type safe enumeration for parameter modes.

Holder Classes

Holder classes enable the mapping to preserve the intended operation signature and parameter passing semantics. A holder class is simply a class that contains an instance of its type. According to the JAX-RPC specification, each Holder class provides the following methods and fields

- A public field named value. The type of value is the mapped Java type
- A default constructor that initializes the value fields to a default value
- A constructor that sets the value field to the passed parameter

For example, the holder for the Double class would be:

```
public final class DoubleHolder imple-
ments Holder {
public double value;
public DoubleHolder() {  }
public DoubleHolder(double  value) {
this.value = value;
}
}
```

In this article, I've used JAX-RPC Synchronous Request-Response Mode to explain the following parameter modes using Apache Axis.

- ***IN*** parameters using simple types
- ***IN*** and ***OUT*** parameters using simple types
- ***INOUT*** parameters using simple types
- ***OUT*** parameters using complex types

IN Parameters Using Simple Types

This example deals with developing a currency converter SOAP service (conversion from dollars to rupees). The service takes only one input parameter of simple type and returns a simple type.

Developing the SOAP Service

For example, develop a currency converter SOAP service as:

```
 package com.mydomain.SampleService;
 public class SampleService {
  public double convertcurrencyin(dou-
ble in) {
       double res = in * 48;
       return res;
  }
 }
```

Here I've used the default rate; it should be replaced by the actual value.

Deploying the SOAP Service

Create a Web Services Deployment Descriptor file (deploy.wsdd) in order to deploy the Web service (see Listing 1). Deploy the service using the deploy.wsdd file as described in the Axis documentation.

Developing the SOAP Client

Create a service using ServiceFactory in order to create the Call objects.

```
Service  service = new Service();
Call call    = (Call)
service.createCall ( );
String endpointURL = http://local-
host:8080/axis/servlet/AxisServlet;
call.setTargetEndpointAddress( new
java.net.URL(endpointURL) );
```

After creating a Call object, configure the call object with the operation (method) name, parameter types, and parameter modes like IN or OUT or INOUT and return type (see Listing 2).

After setting the parameter modes, parameter types, and return types, invoke the target end point by passing the parameters and get the result. The *invoke( )* method calls a specified operation using the *synchronous request-response* interaction mode. Use the *invoke OneWay( )* method for using the *one-way interaction mode*. The invoke method blocks until the remote service receives the method call and return either a response or an exception, whereas invokeOneWay( ) doesn't block and this method is not allowed to propagate a remote exception to the client.

```
Double db = new Double (4.0);
Object ret = call.invoke( new Object[]
{ db} );
```

Use the Axis TCP Monitor tool to see the request/response message between the SOAP server and the client on the wire. The SOAP Messages look like Listing 3.

Observe the request and response message and notice that Response is appended to the name of operation i.e. convertcurrencyin. Apache Axis appends operation/method name for the return element where as some SOAP servers don't. (See source code for this article, available at www.sys-con.com/webservices/sourcec. cfm.)

IN and OUT Parameters Using Simple Types

This example shows how to develop a service and client using in and out parameters. The out parameters are achieved using Holder classes.

Developing the SOAP Service

In this example the client sends the currency to be converted as an input parameter to the service and the client receives the converted currency as output by using the holder classes. The service looks like:

```
 package com.mydomain.SampleService;
 public class SampleService {
public int convertcurrencyout(double
in, DoubleHolder dh) {
       double res = in * 48;
       dh.value = res;
       return 1;
}
  }
```

Deploying the SOAP Service

Add the following entries to the deploy.wsdd file as a child element of the service element and deploy the service.

```
<operation name="convertcurrencyout">
    <parameter name="arg1" mode="IN"/>
    <parameter name="result"
mode="OUT"/>
  </operation>
```

A parameter element is used to specify parameter modes and parameter names. This is one of the ways that Axis SOAP server gets the information about the parameters.

Developing the SOAP Client

Create a Call object and configure the objects with the service details (see Listing 4). Parameters of the type OUT need not be passed to the invoke method.

In order to get the output parameters

from the target end point service, use the method *getOutputParams( )* on the Call object as shown in Listing 5.

The messages in Listing 6 are sent and received from the SOAP server. Observe the SOAP response, it has an additional element "result" after the return element. Each output parameter will have an element in the SOAP response with the output value.

INOUT Parameters Using Simple Types

SOAP INOUT parameters are achieved using Holder classes.

Developing the SOAP Service

In this example the Client sends the currency as an instance of DoubleHolder. The service accesses the currency by using the holder's value field and setting the converted currency in the same field.

The service looks like:

```
  package com.mydomain;
import javax.xml.rpc.holders.*;
public class SampleService {
  public int
convertcurrencyinout(DoubleHolder dh) {
        dh.value  = dh.value * 48;
        return 1;
   }
  }
```

Developing the SOAP Client

Create the Call object and configure it with the service details in Listing 7.

Observe that code and notice that the parameter mode is ParameterMode.INOUT and the passed parameter is the holder's wrapper class, i.e., Double. There is no difference in getting the output from the service for output parameters and INOUT parameters. On the wire the SOAP message looks like Listing 8.

So far I've discussed simple data types only. What about complex data types? JAX-RPC specifies the semantics for Java serialization in terms of XML mapping. Axis provides default serialization and deserialization for the JavaBeans. In my next example I've used user-defined holders (AddressHolder) for the output parameter and user-defined type (Address) and used axis default bean serialization. Use Apache's Java to WSDL tool to generate a WSDL file. It generates complex data types for the user-defined types like Address Object. I have not used WSDL to invoke the remote service

OUT Parameters Using Complex Types

In this example the client sends an order ID to the remote SOAP service and the service will return the shipment address as output parameter in the response message. Since Shipment Address is an output parameter, create a holder class to hold the Address and create Address class according to the JavaBeans Specification.

"Apache Axis appends operation/method name for the return element where as some SOAP servers don'tdone?"

SOAP Service

```
 package com.mydomain;
 public class SampleService {{
 public String getShipmentDetail(String
orderId, AddressHolder adrh) {
     Address adr = new Address();
     adr.setstreet("CunninghamRoad");
adr.setcountry("INDIA");
 adrh.value = adr;
 return orderId;
 }
```

Add the following entries in the deploy.wsdd file inside the service element and register the bean mappings as shown below.

```
<operation name="getShipmentDetail">
    <parameter name="orderId"
mode="IN"/>
    <parameter name="result"
mode="OUT"/>
  </operation>
<beanMapping qname="myNS:Address"
xmlns:myNS="urn:MyService"

languageSpecificType="java:com.mydo-
main.Address"/>
```

SOAP Client

On the client side, register the serializers and deserializers for the user-defined data types, i.e., for the Address object.

```
 QName qn = new QName(
"urn:SampleService", "Address" );
 call.registerTypeMapping(Address.class,
qn,
 new
org.apache.axis.encoding.ser.BeanSeriali
zerFactory(Address.class, qn),
 new
org.apache.axis.encoding.ser.BeanDeseria
lizerFactory(Address.class, qn));
```

Once the type mappings are registered, pass the parameters and invoke the service as shown:

```
 call.setOperationName( new
QName("SampleService",
"getShipmentDetail") );
 call.addParameter( "orderId",
XMLType.XSD_STRING, ParameterMode.IN);
 call.addParameter(
"result",qn,ParameterMode.OUT);
 call.setReturnType(
org.apache.axis.encoding.XMLType.XSD_STR
ING );
String orderId = "ORD_001";
 String val = (String)call.invoke( new
Object[] {orderId} );
```

Retrieving the output parameter is the same as for the simple types. Generating the SOAP message on the wire looks like Listing 9.

If you want to send a user-defined Holder (AddressHolder) as an INOUT parameter to the remote SOAP service, then pass the user-defined type (Address) as an INOUT parameter and register the type mappings.

References

- *Apache Axis:* http://xml.apache.org/axis
- *JAX-RPC:* http://java.sun.com/xml/jaxrpc
- *SOAP:* www.w3.org/TR/SOAP

Listing 1: Web Services Deployment Descriptor file

```
<deployment xmlns= "http://xml.apache.org/axis/wsdd/"
   xmlns:java
="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="SampleService" provider="java:RPC">
   <parameter name="className"
value="com.mydomain.SampleService"/>
<parameter name="allowedMethods" value="*"/>
</service>
</deployment>
```

Listing 2: Configure the call object

```
Call.setOperationName( new Qname("SampleService", "con-
vertcurrencyin") );
call.addParameter( "currency", XMLType.XSD_DOUBLE,
ParameterMode.IN);
call.setReturnType(XMLType. XSD_DOUBLE);
```

Listing 3: SOAP Message

```
SOAP Message request
 <soapenv:Body>
 <convertcurrencyin
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
 <currency xsi:type="xsd:double">4.0</currency>
 </convertcurrencyin>
 </soapenv: Body>

SOAP Message response
<soapenv:Body>
  <convertcurrencyinResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
 <convertcurrencyinReturn
xsi:type="xsd:double">192.0</convertcurrencyinReturn>
</convertcurrencyinResponse>
 </soapenv:Body>
```

Listing 4: Service details

```
             call.setOperationName( new
QName("SampleService", "convertcurrencyout") );
             call.addParameter( "currency", XMLType.XSD_DOU-
BLE, ParameterMode.IN);
             call.addParameter(
"result",XMLType.XSD_DOUBLE,ParameterMode.OUT);
             call.setReturnType(
org.apache.axis.encoding.XMLType.XSD_INT);

             Double db = new Double(4.0);
             Object ret =  call.invoke( new Object[] { db} )
```

Listing 5: Use getOutputParams()

```
Map outparams = call.getOutputParams();
Setse = outparams.keySet();
Object obj[] = se.toArray();       // iterate this array get
the parameter names

// To get the actual value from the service
Collection se1 = outparams.values();
Object obj1[] = se1.toArray(); // // iterate this array get
the parameter values
```

Listing 6: Messages sent and received from SOAP server

```
SOAP Message request
 <soapenv:Body>
 <convertcurrencyout
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
 <currency xsi:type="xsd:double">4.0</currency>
 </convertcurrencyout>
 </soapenv:Body>

SOAP Message Response
 <soapenv:Body>
  <convertcurrencyoutResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
   <convertcurrencyoutReturn
xsi:type="xsd:int">1</convertcurrencyoutReturn>
   <result xsi:type="xsd:double">192.0</result>
  </convertcurrencyoutResponse>
 </soapenv:Body>
```

Listing 7: Service details

```
 call.setOperationName( new QName("SampleService", "con-
vertcurrencyinout") );
 call.addParameter( "currency", XMLType.XSD_DOUBLE,
ParameterMode.INOUT);
 call.setReturnType(
org.apache.axis.encoding.XMLType.XSD_INT);

Double dbinout = new Double(5.0);
 call.invoke( new Object[] {dbinout} );
```

Listing 8:

```
SOAP Message request
<soapenv:Body>
  <convertcurrencyinout
soapenv:encodingStyle ="http://schemas.xmlsoap.org/soap/encod-
ing/" >
   <currency xsi:type="xsd:double">4.0</currency>
  </convertcurrencyinout>
 </soapenv:Body>

SOAP Message Response

<soapenv:Body>
  <convertcurrencyinoutResponse
soapenv:encodingStyle =
"http://schemas.xmlsoap.org/soap/encoding/" >
```

```
    <convertcurrencyinoutReturn
xsi:type="xsd:int">1</convertcurrencyinoutReturn>
    <currency xsi:type="xsd:double">192.0</currency>
   </convertcurrencyinoutResponse>
  </soapenv:Body>
```

Listing 9: SOAP Message request

```
<soapenv:Body>
  <getShipmentDetail
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
    <orderId xsi:type="xsd:string">ORD_001</orderId>
   <getShipmentDetail>
  </soapenv:Body>



SOAP Message response

<soapenv:Body>
   <ns1:getShipmentDetailResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/"
   xmlns:ns1="SampleService">
    <getShipmentDetailReturn
xsi:type="xsd:string">ORD_001</getShipmentDetailReturn>
    <result href="#id0"/>
   </ns1:getShipmentDetailResponse>
   <multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/"
   xsi:type="ns2:Address" xmlns:soapenc="http://schemas.xml-
soap.org/soap/encoding/"
   xmlns:ns2="urn:SampleService">
    <country xsi:type="xsd:string">INDIA</country>
    <street xsi:type="xsd:string">CunninghamRoad</street>
   </multiRef>
  </soapenv:Body>
```

Download the code at
sys-con.com/webservices

SHOP ONLINE AT **JDJSTORE.COM** FOR BEST PRICES OR CALL YOUR ORDER IN AT **1-888-303-5282**

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

GUARANTEED BEST PRICES
FOR ALL YOUR
WEB SERVICES
SOFTWARE NEEDS

JDJStore.COM

N-ARY
$299.00 — n-ary Ticket System v2.0

This installable application is a clean-cut Web-based tool that enables you to log & track important information. It can be used as an intranet or extranet product, giving your customers access to see how their issue is progressing. A unique number ID is assigned to every ticket. The system is extremely flexible and simple to use and can be utilized in numerous different situations. With more features than before, The Ticket System can be used for any number of great applications, which you can tailor as much or as little as you like.

GREENPOINT INC.
$1,350.00 — WEB CHARTS 3D VER. 4.7

WebCharts 3D is a state-of-the-art visualization package designed for the professional Web developer. It provides general purpose and specialized 2- and 3-dimensional charts, grids, and heat maps that can be delivered as server-generated interactive images (PNG, GIF, JPG, SWF, SVG, WBMP) or applets to browsers and mobile devices.

CHUTNEY TECHNOLOGIES
$755.25 — Chutney SOAP+ Toolkit

The Chutney SOAP+ Toolkit is the industry's first Web services monitoring and optimization toolkit. The Toolkit fills the functionality gaps in the leading SOAP development libraries by providing the ability to accurately pinpoint Web services bottlenecks and eliminate tthem through optimization techniques such as caching. The Chutney SOAP+ Toolkit acts purely as a supplement to these libraries, so no changes to the existing application logic are required. Flexible in its feature set, the Toolkit provides value to applications serving as either Web service consumers or providers.

ALTOWEB
$2,500.00 — Application Platform Release 2.8

The AltoWeb Application Platform lets you build, deploy, and manage J2EE applications and Web services up to 10x faster without requiring extensive J2EE or Web services expertise. How? By replacing lengthy, custom, and complex J2EE, XML, and Web services coding with rapid component assembly and reuse.

ZION
$895.00 — JMessageServer(TM) Workgroup (up to 50 users)

An open Instant Messaging & Presence Server written in Java, with a Web-based account manager and database buddy list and presence. Zion supports JSuperChat IM clients and custom clients enabled with JBuddy SDK for real-time communications and Enterprise Application Integration (EAI)

POINTBASE
$353 .00 — PointBase Server 4.4 Windows Installer

PointBase® Server is a 100% Pure Java RDBMS for client-server networked applications. PointBase Server provides a reliable and scalable database for workgroup servers, application servers, web servers and other Internet-based applications needing storage of metadata.

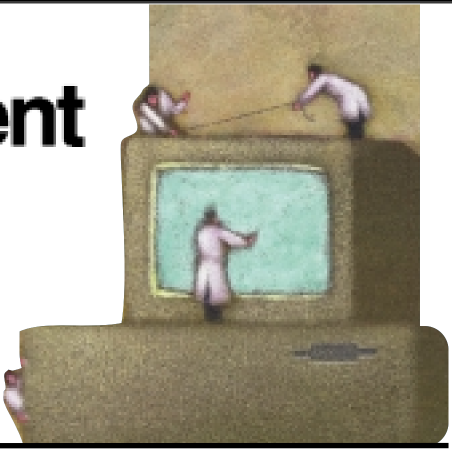
W W W . J D J S T O R E . C O M

OFFERS SUBJECT TO CHANGE WITHOUT NOTICE

Written by Jinyou Zhu

Web Services Development with PowerBuilder 9

Build Web service clients in both Windows and JSP applications



PowerBuilder, a 4GL RAD tool, has extended its productivity to Web services components and application development. With this RAD environment, developers may not only make use of Web services, creating new Web services components and applications, but also easily migrate existing components to Web services. This article discusses how to consume different kinds of Web services in PowerBuilder applications and how to produce PowerBuilder components as Web services in PowerBuilder 9.

AUTHOR BIO:

Jinyou Zhu is a senior software engineer with Sybase, Inc., and a leading developer of database and Web services for PowerBuilder 9. His expertise includes database, XML parser, Web services, and multilanguage support.

Introduction

Web services are loosely defined as the use of Internet technologies to make distributed software programs talk to each other. Programs that want to interact with one another over the Web must be able to find one another. They also must figure out what the expected interaction patterns are, such as what methods the services offer, what data types the services accept, what protocols they can use to communicate with each other, and the quality of services such as security, transaction, and reliable messaging. Then they must have an agreement on messaging systems so they can understand each other.

To meet these requirements, W3C has developed a collection of standards such as Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery, and Integration (UDDI). With these specifications, developers may create and consume Web services as well. However, these specifications are too complex for application developers. PowerBuilder abstracts the complexity and provides a simplified and productive 4GL development environment for developers to use instead.

Building a Web Services Client

PowerBuilder has always been an open development environment for building client-side applications. PowerBuilder 9 allows developers to build a generic client to consume Web services in Windows applications and Web applications. They can make use of any Web services no matter if the services are created in PowerBuilder, .NET, Java, or other third-party tools. In the next two sections, we'll show you how to easily consume Web services in these two kinds of applications. We'll use http://www. xmethods.net/sd/2001/CurrencyExchangeService.wsdl for the example.

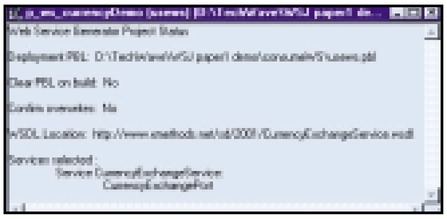
Consuming Web Services in a Windows Application

To consume Web services in PowerBuilder Windows applications, you first need to create a Web Service proxy, which represents the remote Web service. Then you may call the Web service via the proxy in the application. The following steps provide more details.

Step 1: Use Web Service Proxy Wizard to Create a Proxy from WSDL File

In PowerBuilder 9, the Web Service Proxy Wizard helps developers generate the proxy for developing an application. To start the wizard, select the Web Service Proxy Wizard icon from the Project page of the New dialog box. Following the wizard:

- ***Specify WSDL file:*** It may be a local file or a remote file from the Internet. In this sample, we use www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl.
- ***Select the service from the service list within the WSDL file.***
- ***Choose the port or ports to use:*** Only SOAP ports are supported in PowerBuilder 9. So selecting ports with other type (such as Get, Post) will result in error.
- ***Give a prefix that is appended to the port name:*** It becomes the proxy name.
- ***Specify the PowerBuilder library where the proxy will be deployed.***



[illegible]

Web Service Generator Project Status

Deployment PBL: D:\Techwave\WSJ paper\ demo\consumeWS\usews.pbl

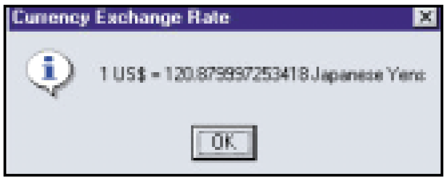
Clear PBL on build: No

Confirm overwrites: No

WSDL Location: http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl

Services selected :
Service CurrencyExchangeService
CurrencyExchangePort

FIGURE 1 Proxy Project for Currency Exchange Service



Currency Exchange Rate

1 US$ = 120.879997253418 Japanese Yens

FIGURE 2 Result

- ***Give a project name:*** It will store the information given above and deploy the proxy.

Once finished, the wizard will create the project into its current PowerBuilder library. The project looks like Figure 1. Right-click the project in the system tree and select Deploy. The project will generate the proxy automatically. If there are complex types in the WSDL file, the project will generate PB structures for them too. Now the proxy is ready for PowerScript.

Step 2: Use the Proxy to Call Web Services

Invoking Web services through SOAP requires serialization and deserialization of data type, building SOAP messages, and parsing SOAP messages. In PowerBuilder 9, PBSoapClient90.pbd and PBSoapClient90.dll are developed for this task. These two files are installed in the Shared/PowerBuilder directory when PowerBuilder 9 is installed. The PBSoapClient90.pbd must be added into the applications and PBSoapClient90.dll must be deployed with your application executables. PowerBuilder 9 (GA release) will add PBSoapClient90.pbd into the system tree automatically when you create an application.

If you're using a beta version, you may need to do this manually. To do so, right-click the application in the system tree and select Properties, then type in the full path and name in the Library List or browse the file from the hard disk.

There are two classes in PBSoapClient90.pbd, namely SoapConnection and SoapException. The SoapConnection class is used to instantiate the proxy object, populate the SOAP options, and connect to the SOAP server. It has three public methods that we can use immediately.

- ***CreateInstance(proxy_object, proxy_classname):*** Create proxy instance using default endpoint, which is stored in the proxy.
- ***CreateInstance(proxy_object, proxy_classname, endpoint):*** Create proxy instance using the endpoint specified. The default endpoint will be ignored.
- ***SetOptions(optionString):*** Set options for the SoapConnection. There are three options available: SoapLog, UserID, and Password. The string values for the option names are case insensitive. The format for option string is name/value pairs. For example, the option string could be:

```
"SoapLog=~"Logpath~", userID=~"user-
name~", Password=~"password~"
```

The SoapException class is used to process exception. It inherits from the PowerBuilder Runtime error class. All exceptions and errors during the execution of Web services will be converted to Soap Exception and thrown to calling script. PowerScript can use try-catch black to capture this exception.

The script in Listing 1 shows you how to invoke Web services in PowerScript. It creates an instance of SoapConnection and sets the options for the connection first. In the example, we only set the SOAP log file to capture the exchanged messages between PowerBuilder client and SOAP server. Then it creates an instance for the proxy. Next, it invokes the method ("get Rate") of the Web service and prints out the result. If anything goes wrong, the script will capture the exception and print out an error message.

Result

Now we can run our application; we can see a message box with the correct return value (see Figure 2).

It's really simple to build PowerBuilder applications to consume Web services. The sample Web service used here is developed in Java. And we can make use of it in our PowerScript. Besides that, PowerBuilder can also consume Web services that are developed in any other language and run on any Web servers. such as .NET server, Apache, WebLogic, WebSphere, and EAServer. Visit www.xmethods.com for more information or to create your own Web services in PowerBuilder 9.

Consuming Web services in a JSP Application

PowerBuilder 9 enables developers to build and use JavaServer Pages (JSPs) in their PowerBuilder Web applications. A JSP application in PowerBuilder 9 may consume Web services as well. Create a JSP target with the source and build folder and deployment configuration first. The JSP may be deployed to EAServer or Tomcat. In this example, we use EAServer. Then we use the JSP Web Service Proxy Wizard on the Web page of the New dialog box to create a custom tag library with the information necessary for calling a Web service in a JSP. The JSP Web Service Proxy Wizard is similar to the Web Service Proxy Wizard for the PowerBuilder Windows application. It collects information such as the location of a WSDL file, the service, and port. We can specify overrides to the WSDL file for a custom bean name, Java class name, Java package name, TLD name, JAR file name, output variables, and the selection of operations within a service. The wizard creates a TLD file, the Java source code to process the custom tag, and the compiled Java class files of the source.

Once the custom tag is generated, it appears in the system tree. To use custom tags to invoke Web services, do the following:

- Declare the custom tag library to the JSP page by dragging the TLD file from the Components tab of the system tree to the Page tab of the JSP page. We use "sg" as the prefix. Once it is dragged to the JSP page, the tag library is automatically associated with the page.

- Declare input arguments for the custom tag in server script if needed.
- Declare custom tag of Web services to the JSP page by dragging the method of the Web service from the Components tab of the system tree to the Page tab of the JSP page. Specify the input arguments for the custom tags for the Web services in the JSP. We use "US" as country 1 and "Japan" as country 2.
- Output arguments are stored in the page Context of the JSP container. Its name is "returnValue".
- Set the properties of the JSP target for deployment and run.
- Build the JSP pages and deploy WAR files to the EAServer. To do so, right-click the JSP target and select Deploy.

The TLD file for the currency exchange example used earlier looks like Listing 2. The source code of the JSP page to call the Web service is shown in Listing 3. Now run the JSP page in PowerBuilder or browse to the JSP page just deployed. If all went well, the correct exchange rate will show on the page. If anything goes wrong, consult the log file.

A custom tag for Web services throws a JspTagException for non-recoverable errors if something goes wrong. The JspTag Exception contains information about the root cause of the exception and the point where the error occurred in processing the custom tag. This exception can be caught in a Try-Catch block or mapped to a specific error page in the Deployment Configuration Properties dialog box for the JSP target.

Note: JSP in PowerBuilder 9 now supports only RPC-style Web services.

Deploying Web Services in PowerBuilder 9 to EAServer

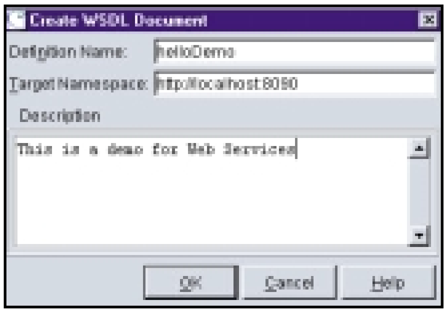
PowerBuilder, as a 4GL RAD tool, extends to component and Web service development as well. I'll use EAServer 4.1.3 as both application server and Web server. What follows is a five-step program for developing EAServer components and deploying them to Web services via EAServer. If you already have your PowerBuilder components, you may skip Step 1.

Step 1: Develop EAServer Component in PowerBuilder 9

In PowerBuilder, there are several kinds of wizards – such as application, template application, EAServer component, COM/MTS component, and automation server – to help developers generate PowerBuilder applications. In our case, we chose the EAServer component. Before running the wizard to generate the EAServer component, the EAServer must be started and the EAServer profile must be created in PowerBuilder.

We selected the EAServer Component Wizard icon from the Target page of the New dialog box. As you work through the wizard, you'll make the following decisions:

- ***Application name:*** We call it "hello". The PB library and PB target with the same name will be generated automatically.
- ***Interface Options:*** We used new interface.
- ***PowerBuilder object name:*** We used "n_hello".
- ***Component name:*** To be the component name when deployed to EAServer.
- ***Select the EAServer you want to use***.
- ***Package name:*** The package name when deployed to the EAServer. We use "p_hello".
- ***Component type:*** We use standard type.
- ***Determine if the instance pooling will be used***.
- ***Determine if the component supports transaction***


Create WSDL Document

Definition Name: helloDemo

Target Namespace: http://localhost:8080

Description

This is a demo for Web Services

FIGURE 3 **Create WSDL documents**

- ***Select interface options***.
- ***Project name:*** This project will be used to deploy the component.

For most wizard pages, we use the default values. Once finished, you'll see a PB target named "hello" within the system tree. Under the target, a project ("p_hello_eascomps") and a component ("n_hello") were already created. Now you may develop any functions in the component. The following function is an example:

```
function string f_hello (string name)
    return string("Hello, Web services
from") + string(name)
end function
```

Step 2: Deploy PowerBuilder Object to EAServer

Once the development of the component is finished, right-click the project ("p_hello_eascomps") and select Deploy to deploy the component to EAServer.

Step 3: Wrap the PowerBuilder Component to a J2EE Component

In Jaguar manager, an EAServer management console, you need to generate a client-side stub and server skeleton for the deployed package/component. To generate the stub/skeleton, right-click the just deployed package, then select Generate Stub/Skeleton. On the dialog, select "Cobra" as the protocol and generate and compile Java files for stub/skeleton.

Step 4: Create WSDL Documents and Services

Web Services Toolkit (WST) will wrap the components to SOAP components and generate WSDL files. WST generates two WSDL files for each component — one is for the interface of the service and the other is for implementing information of the service.

To generate WSDL files, first right-click the

WSDL Documents folder and select New WSDL Document. Figure 3 shows the dialog for Create WSDL Document. Give the definition name ("helloDemo") and target namespace. The target namespace not only works as the "TargetNamespace" attribute in the WSDL file, but also specifies the location where the implementation WSDL try to look for the interface WSDL file. So it's critical for WST to generate a workable Web service. Give a URL where you want to put the interface WSDL file. Secondly, create a new Web service by right-clicking "hello Demo" document and select New Web service. You can browse all EAServer components and select any one to use. For this article, we selected package "p_hello" and component "n_ hello". Finally, add properties to the Web service. You need to specify the address for SOAP listener and which functions in the component to be exposed to Web services.

Step 5: Copy the WSDL Files For Use

The two WSDL files generated at the last step reside under <EAServer host folder>\WebServices\work\wsdl\. Copy them to the virtual directory of the Web server. In this tutorial, we copied them to the virtual directory of http://localhost: 8090. Now the Web services are ready for use.

Summary

PowerBuilder 9 abstracts the complexity of the Web services development and provides a wizard-guided development environment. The sample applications in this article demonstrate the ease of Web services development. PowerBuilder 9.0 is an excellent development tool to build Web service clients both in Windows and JSP applications, and produce Web service components.

Listing 1: Calling Web Service from PowerScript

```
long l_ret
real r_rate

SoapConnection conn
demo_currencyexchangeport p_obj

//Create connection instance
conn =  create SoapConnection

//Set options for the connection
conn.SetOptions("SoapLog=~"c:\\soaplog.txt~"")

// Create proxy instance
l_ret = Conn.CreateInstance(p_obj, "demo_currencyexchange-
port")
if l_ret <> 0 then
    MessageBox("Error", "Cannot create instance of proxy")
    return
end if

// Invoke Web Service
try
    r_rate = p_obj.GetRate("us","japan")
    MessageBox("Currency Exchange Rate","1 US$ = "+
string(r_rate) + " Japanese Yens")
catch ( SoapException e )
    MessageBox ("Error", "Cannot invoke Web Service~n"
+e.getMessage())
end try

destroy conn
```

Listing 2: TLD File for currency exchange Web service

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP
Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>CurrencyExchangeService</shortname>
    <tag>
        <name>getRate</name>
        <tagclass>CurrencyExchangeService.getRate</tag-
class>

<teiclass>CurrencyExchangeService.getRate_tei</teiclass>
        <bodycontent>EMPTY</bodycontent>
        <attribute>
            <name>country1</name>
            <required>yes</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>country2</name>
            <required>yes</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

Listing 3: Call Web Service from JSP application

```
<%@ taglib prefix="sg" uri="WEB-
INF/tlds/CurrencyExchangeService.tld" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
    <META HTTP-EQUIV="PowerSiteData" NAME="SERVERLANGUAGE"
CONTENT="Java">
    <TITLE>DemoPage</TITLE>
    <META http-equiv="Content-Type" content="text/html;
charset=windows-1252">
    <META content="PB9.0.0.5001" name="GENERATOR">
</HEAD>
<BODY bgColor="white" PSPARAMS="">
<P>1US$ = &nbsp;</P>
<P><sg:getRate country1="us" country2="japan" /></P>
<P><%= CurrencyExchangeService_getRate_returnValue %></P>
<P>Japanese Yens &nbsp;</P>
</BODY>
</HTML>
```

Download the code at
sys-con.com/webservices

A LIMITED TIME SAVINGS

SUBSCRIBE TODAY TO MULTI

Go To www.sys-con.com

and receive your **FREE** CD Gift Package VIA Priority Mail

Each CD is an invaluable developer resource packed with important articles and useful source code!

Pick the CDs to go with your Multi-Pack order

Pick one CD with your 3-Pack order

Pick two CDs with your 6-Pack order

Pick three CDs with your 9-Pack order

☐ Web Services Resource CD

☐ Java Resource CD

☐ WebLogic Resource CD

☐ ColdFusion Resource CD

☐ XML Resource CD

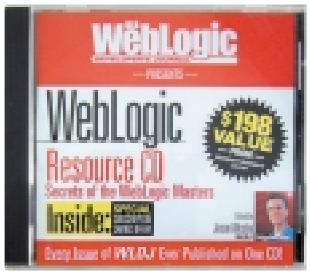
☐ WebSphere Resource CD

☐ CF Advisor Complete Works CD

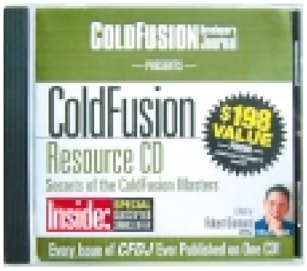
Your order will be processed the same day!

More than 1,400 Web services and Java articles on one CD! Edited by well-known editors-in-chief Sean Rhody and Alan Williamson, these articles are organized into more than 50 chapters on UDDI, distributed computing, e-business, applets, SOAP, and many other topics. Plus, editorials, interviews, tips and techniques!
LIST PRICE $198

The most complete library of exclusive *JDJ* articles compiled on one CD! Assembled by *JDJ* Editor-in-Chief Alan Williamson, more than 1,400 exclusive articles are organized into over 50 chapters, including fundamentals, applets, advanced Java topics, Swing, security, wireless Java,... and much more!
LIST PRICE $198

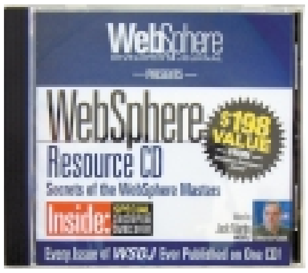


The most complete library of exclusive *WLDJ* articles ever assembled! More than 200 articles provide invaluable information on "everything WebLogic", including WebLogic Server, WebLogic Portal, WebLogic Platform, WebLogic Workshop, Web services, security, migration, integration, performance, training...
LIST PRICE $198



The most complete library of exclusive *CFDJ* articles on one CD! This CD, edited by *CFDJ* editor-in-chief Robert Diamond, is organized into more than 30 chapters with more than 400 exclusive articles on CF applications, custom tags, database, e-commerce, Spectra, enterprise CF, error handling, WDDX... and more!
LIST PRICE $198

The largest and most complete library of exclusive *XML-Journal* articles compiled on one CD! Edited by well-known editors-in-chief Ajit Sagar and John Evdemon, these articles are organized into more than 30 chapters containing more than 1,150 articles on Java & XML, XML & XSLT, <e-BizML>, data transition... and more!
LIST PRICE $198



The most up-to-date collection of exclusive *WSDJ* articles! More than 200 articles offer insights into all areas of WebSphere, including Portal, components, integration, tools, hardware, management, sites, wireless programming, best practices, migration...
LIST PRICE $198

The complete works of *CF Advisor* are now on one CD! Check out over 200 exclusive articles on topics such as e-commerce, custom tags, Fusebox, databases, object-oriented CF, WDS, upgrading CF, wireless, Verity, and more. Plus, find interviews, editorials, and source code!
LIST PRICE $198

Subscribe Online Today

OFFER FROM SYS-CON MEDIA

PLE MAGAZINES ONLINE

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

AND SAVE UP TO $400 AND RECEIVE UP TO 3 FREE CDs!

Pick a 3-Pack, a 6-Pack or a 9-Pack ✓

TO ORDER: Choose the Multi-Pack you want to order by checking next to it below. Check the number of years you want to order. Indicate your location by checking either U.S., Canada/Mexico or International. Then choose which magazines you want to include with your Multi-Pack order.

☐ 3-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.
☐ 6-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.
☐ 9-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.

☐ **Java Developer's Journal**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Web Services Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **.NET Developer's Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **XML-Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebLogic Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **ColdFusion Developer's Journal**

U.S. - Two Years (24) Cover: $216	You Pay: $129 /	Save: $87 + FREE $198 CD
U.S. - One Year (12) Cover: $108	You Pay: $89.99 /	Save: $18
Can/Mex - Two Years (24) $240	You Pay: $159.99 /	Save: $80 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $99.99 /	Save: $20
Int'l - Two Years (24) $264	You Pay: $189 /	Save: $75 + FREE $198 CD
Int'l - One Year (12) $132	You Pay: $129.99 /	Save: $2

☐ **Wireless Business & Technology**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $192	You Pay: $139 /	Save: $53 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $79.99 /	Save: $16
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebSphere Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **PowerBuilder Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

3-Pack

Pick any 3 of our magazines and save up to $275.00
Pay only $175 for a 1 year subscription plus a **FREE CD**

- 2 Year – $299.00
- Can/Mex – $245.00
- International – $315.00

6-Pack

Pick any 6 of our magazines and save up to $350.00
Pay only $395 for a 1 year subscription plus **2 FREE CDs**

- 2 Year – $669.00
- Can/Mex – $555.00
- International – $710.00

9-Pack

Pick all 9 of our magazines and save up to $400.00
Pay only $495 for a 1 year subscription plus **3 FREE CDs**

- 2 Year – $839.00
- Can/Mex – $695.00
- International – $890.00

www.sys-con.com/suboffer.cfm

SYS-CON MEDIA

The Symbiosis of Innovation

Building standards-based Java Web services

The next time a CEO or CIO comes to you looking for an innovative solution to an intractable business problem, you might want to remember the example set by the Medusa jellyfish and the snail.

In his essay "The Medusa and the Snail" (from the book of the same name), biologist and author Lewis Thomas demonstrated how innovative forms of cooperation offer a better way for organisms to propagate than simple competition. To make his point, Lewis wrote about the symbiotic relationship between a jellyfish and a snail. At various points in their interconnected life cycles, the jellyfish and snail must feed off each other in order to reproduce. Somehow, over a period of eons, these two species evolved this unique relationship of mutual dependence and mutual benefit to ensure the proliferation of both species.

You can see a similar form of innovative cooperation taking shape today in the realm of Web services development. In this case, it's Java tools and industry standards that have formed a symbiotic relationship. The integration of Java technology and standards is making it easier for developers to create Web services that solve real business problems and innovate new solutions. Furthermore, the ongoing evolution of this relationship will ensure that Web services – and the innovations they make possible – will continue to proliferate and grow.

The importance of standards in Web services is widely recognized. According to a recent survey of CIOs in Europe, their top priority when considering Web services was to find a solution based on standards. This was the most important factor (31%), followed by scalability/extensibility (28%), comprehensiveness of a solution (15%), price (7%), and performance (6%).

Web Services Require Standards

The very notion of Web services is predicated on a model of mutual dependence and mutual benefit. Web services (or services on demand) cannot exist unless all of the applications and services involved are able to communicate and interact fully – not simply exchange data in a common format.

To solve real business problems, applications and Web services must be able to collaborate immediately as a seamless, coordinated team. It's no coincidence that many experts are now using words like "choreography" and "orchestration" to describe the high degree of interaction required between parties in Web services. When the interaction between an application and a Web service is that well coordinated, rich, and seamless, they can collaborate as flawlessly and gracefully as Fred Astaire and Ginger Rogers.

That kind of interaction requires a relationship as full and complex as the one shared by the jellyfish and the snail and is only possible through the continuing development and evolution of industry standards, from XML to WSDL to SOAP to WSCI.

By supplying the language, syntax, business registries, message protocol, and message flow descriptions of Web services, these and other standards improve the meaningfulness and quality of Web services interaction.

Web services require open development tools. While standards bodies do the job of establishing the rules and means of Web services interaction, they cannot ensure that Web services will be available on demand, to anyone, anywhere. That can only happen if developers use nonproprietary tools that support the development of open, portable, interoperable, and extensible Web services.

For many developers, that has meant Java tools. While Sun Microsystems invented the Java programming language and pioneered its use for enterprise services, the Java 2 Enterprise Edition (J2EE) standard represents a collaboration among leaders from throughout the enterprise software arena. Sun's partners include OS and database management system providers, middleware and tool vendors, and vertical market applications and component developers. Working with these partners, Sun has defined a portable and open platform that can be implemented on an array of enterprise systems and that supports the deployment of Web services.

Developers seeking to build standards-based Java Web services can choose from a number of integrated development environments (IDEs) from a variety of other vendors.

The Evolution of Standards from Data Exchange to Collaboration

Like the jellyfish and the snail, standards and Java tools can be seen as living organisms,

Author Bio

Mike Bellissimo is senior director of Sun Software Developer Marketing and Management. He is a Sun veteran who has held positions managing Sales Operations for iPlanet and JavaSoft, and Software Training and Services for SunSoft. He holds a BA in communications from the University of Pennsylvania and an MBA in high-technology management from Northeastern University.
MIKE.BELLISSIMO@SUN.COM

and as in nature, symbiosis is an ongoing process, not a permanent state of being. To continue delivering benefits to users and enable further innovation, both standards and Java tools must constantly evolve in response to their environment, their needs, and each other.

The earliest and most important standard in the development of Web services was XML. As the standard for exchanging data on the Internet, XML has spawned its own growing family of follow-on standards, including SOAP, WSDL, UDDI, and ebXML.

These standards have enriched the interaction between applications and Web services, helping their relationships evolve beyond simple data exchange toward more problem-solving needs, such as describing business processes for more successful B2B collaboration.

Indeed, business collaboration – another relationship analogous to that of the Medusa and the snail – is a growing focus for Web services standards. Take WSCI, for example. WSCI builds upon current Web service technologies by enabling users of a service – such as another Web service or an application – to understand how to interact with it in a way that is meaningful to a greater collaboration. As a result, developers and architects will be better able to design and create collaborative business processes based on the Web service model.

WSCI also enables developers and architects to describe and compose a global view of the dynamic of the message exchange, another important part of creating a meaningful collaboration. By providing a standard XML syntax for service choreography, WSCI is also essential to ensuring interoperability of Web services.

Understanding their importance to solving real business problems, the Java community has ensured that Java tools support all of these standards.

Forbes.com: A Web Services with Java Success Story

How well does the symbiosis of Java development tools and industry standards solve real business problems? The experience of Forbes.com provides one example.

CEOs, CIOs, COOs, and other key executives depend daily on Forbes.com to deliver up-to-the-minute financial and business news. For Forbes.com – which calls its site the "Home Page for the World's Business Leaders" – the challenge is to continually innovate to provide its executive audience with new and vital information. At the same time, news and financial data are constantly changing, posing an additional challenge of broadcasting the news via the Web in real time.

To meet this challenge, Forbes.com employs over 50 marquee software products, including the J2EE platform. The Forbes.com development team used J2EE to build both its content-management and advertising serving systems. J2EE has also been used to create Forbes.com lists, such as "The Best Cities for Singles."

"We like the rapid development environment that Java technology gives us, because we deploy applications on the site every Thursday," said Mike Smith, chief technology officer for Forbes.com. "We make innovations to the site constantly, and do custom work for advertisers, such as targeting ads demographically and contextually. Java technology's quick deployment capabilities allow us to turn things around overnight. The scalability of J2EE is also critical. It enables us to grow without worrying about overtaxing the system."

Forbes.com also uses Jitzu, from Information Architects, to create Web services. Jitzu, built with the Sun ONE Studio IDE, manages complex components via the Web. Jitzu uses UDDI and SOAP to initiate conversations with a UDDI service. SOAP simplifies remote object access by allowing applications to invoke object methods, or functions, residing on remote servers. A SOAP application creates a request block in XML, supplying the data needed by the remote method, as well as the location of the remote object itself. WSDL (Web Service Description Language) messages are embedded within the SOAP envelope.

Through the use of these Web service protocols and standards, the Jitzu portal server can be located anywhere while allowing complex components such as databases, flat files, or graphics to be downloaded from remote servers in any location worldwide. This all happens transparently to the end user, who simply drags and drops the Web content. This is different from many portals, which require all content to be located on the local application server, thereby severely restricting access to complex components.

"By employing this service-based architecture, Jitzu eliminates the need for us to employ large teams of engineers, Web designers, and developers," said Mr. Smith. "To Web-enable content, we simply set up a component server and register it with a primary server." Once registered, the Jitzu solution downloads the requested complex components to the end user. Forbes.com uses XML to share documents and data from its content repository with its syndication partners. Forbes.com syndicates its content, such as news feeds, to Yahoo, Dow Jones, MSN, Netscape, LexisNexis, and others. These syndication partners also use XML. As a result, Forbes.com can turn syndication implementations around in a day.

The Java Community

Such early successes point to the need to ensure the continuing symbiosis of Java tools and standards, unfettered by proprietary roadblocks or creeping incompatibilities. Fortunately, Java technology benefits from having a large, active user community that has taken major strides to ensure that all applications written in the Java language can communicate fully with each other in the realm of Web services.

The focus of their efforts is the Java Community Process. Since its introduction in 1995 as the open, inclusive process to develop and revise Java technology specifications, reference implementations, and technology compatibility kits, the Java Community Process program has fostered the evolution of the Java platform in cooperation with the international Java developer community. More than 120 Java technology specifications are in development in the Java Community Process program, which has over 350 company and individual participants.

Each Java standard API that is developed through the JCP consists of:

1. A specification that must be approved by the expert group and the Java executive council
2. A reference implementation that proves that all features in the specification can be implemented
3. A Technology Compatibility Kit (TCK) so that all vendors can test their implementations to make sure that they are compatible

Java Specification Requests

New enhancements to Java specifications are submitted as Java Specification Requests (JSRs). A look at three of the most recent JSRs reveals how committed the Java community is to ensuring the openness of Web services. It also provides some ideas about how Java tools will help companies address business problems via Web services in the near future.

JSR 168

JSR 168 defines a set of APIs for portal computing addressing aggregation, personalization, presentation, and security. This will enable interoperability between portlets (Web components similar to servlets) and portals. Developers will be able to write portlets in a way that makes them universally applicable across all portals from all vendors. This will represent a significant step forward in defining portlets as a prime interface to Web services, which can be conveniently aggregated and personalized within portals.

By making portlets and portals more open, JSR 168 will simplify portal deployment for companies, while also allowing third-party developers to create a single, widely deployable portlet, reducing development efforts. Portlets will also be able to interact and share information with each other and Web services, creating a common integration layer below the presentation of the portal itself.

JSR 172

JSR 172 is designed to bring the full benefits of Web services to Java 2 Micro Edition (J2ME), the optimized Java runtime environment for smaller devices such as smart cards, pagers, and cell phones. JSR 172 is designed to provide basic XML processing capabilities; enable reuse of Web service concepts when designing J2ME clients to enterprise services; provide APIs and conventions for programming J2ME clients of enterprise services; adhere to Web service standards and conventions around which the Web services and Java developer community is consolidating; enable interoperability of J2ME clients with Web services; and provide a programming model for J2ME client communication with Web services, consistent with that for other Java clients, such as J2SE.

JSR 198

JSR 198 is a proposal to create a stan-

How Sun's Tools Support Standards and Web Services Development

Sun's tools, including the Sun ONE Studio IDE (formerly Forte for Java) and the Java Web Services Developer Pack, support the development of standards-based Web services.

As part of the Sun Open Net Environment (Sun ONE) framework, (see "The Sun ONE Architecture, Why Care?" *WSJ*, Vol. 2, issue 11) the Sun ONE Studio IDE, based on the open-source NetBeans platform, provides standards-based tools for the Java, C++, C, and Fortran languages. The Sun ONE Studio IDE's modules, wizards, templates, and generators enable development and deployment of J2EE technology–based applications and standards-based Web services to the market-leading application servers, including the Sun ONE Application Server, BEA's WebLogic Server, and, through Extension Partners, to the Oracle9*i* Application Server, IBM WebSphere, JBoss, and more. Its open architecture also facilitates integration with many third-party tools from vendors such as Embarcadero Technologies, Sitraka, Aqris Software, and many others. These vendors have plug-ins that can easily be integrated into the Sun ONE Studio IDE. Vendors can take advantage of the productivity tools provided by the IDE, and customers can take advantage of the added functionality the third-party vendors bring to the IDE.

Sun's Java WSDP

Sun's Java Web Services Developer Pack (Java WSDP) is an integrated toolset available at no charge over the Internet. In conjunction with the Java platform, Java WSDP allows developers to build, test, and deploy XML applications, Web services, and Web applications. The Java WSDP provides Java standard implementations of existing key Web services standards including XML parsing (DOM, SAX, W3C Schema), WSDL, SOAP, ebXML, and UDDI as well as Java standard implementations for Web application development, such as JavaServer Pages (JSP) and the JSP Standard Tag Library. These Java standards are developed through the Java Community Process (JCP), and as such ensure portability and compatibility across vendor implementations.

The Sun ONE Studio IDE incorporates the Java standard APIs included in the Java WSDP, thereby broadening its Web services capabilities. The Sun ONE Studio software implements easy-to-use wizards that utilize the Java APIs for XML and Web services under the hood. For example, it uses the Java API for XML-based RPC (JAX-RPC) to develop SOAP-based Web services and to automatically generate WSDL files that describe the SOAP service. Sun ONE Studio also includes such Java WSDP features as a lightweight UDDI registry, and support for Apache's Tomcat Web server. For Web tier development, Sun ONE Studio includes JSP Standard Tag Library for development of custom tags, and support for multiple client types (Web, mobile, etc.), allowing for different SOAP styles on the client.

dard interface for adding extensions to Java IDEs. This will allow vendors to write an IDE extension once and have it work on any J2SE-compliant IDE.

Web Services Interoperability

Like the jellyfish and the snail, Java tools and industry standards must coexist peacefully, not only with each other, but also with the other "organisms" in the Web services environment. To that end, Sun recently joined the Web Services Interoperability Organization (WS-I) as a contributing member. WS-I is an open industry effort chartered to promote Web services interoperability across platforms, applications, and programming languages. Sun plans to play an active role in WS-I technology guideline areas, including key foundational work currently being developed under the Web Services Basic Profile.

Sun is also a significant participant and contributor to the SOAPBuilders interoperability group. Sun's Web services endpoints are available over the Internet for interoperability testing by anyone, and can be found at http://soapinterop.java.sun.com.

The Future of Java Tools and Web Services

A student of evolution can't help but marvel at the ruthless efficiency and remarkable adaptability of nature. Of course, such adaptations as that between the Medusa and the snail usually take place over a period of millennia. The business world isn't quite so patient.

The Java community and the standards organizations have already demonstrated how a relationship of mutual dependence and mutual benefit can accelerate the evolution of Web services. This ongoing evolution will in turn afford businesses the opportunity to more easily form symbiotic relationships with complementary partners throughout the world, thereby solving many of their most critical problems and ensuring their continued survival.

References

- WSJ-IN News Desk. October 15, 2002. www.sys-con.com/webservices/article-news.cfm?id=380

Techniques for Robust, Database-Driven Web Services

Reuse of existing assets can create new interoperability

Web services have now officially outlived the initial wave of hype, and many corporate developers are in some phase of designing, building, testing, or deploying their first mission-critical services. While Web services offer a new way to expose your application functionality to programmatic clients on different platforms, it's no surprise that they are developed following the same process you use to deliver any high-quality software. This means the life cycle for developing Web services will start with good design and modeling, and include revision control, coding, debugging, and performance profiling, as well as both unit testing and system testing. Adopting a strong toolset that supports this complete Web services and application development life cycle will positively impact the success of your initial rollout and ongoing maintenance.

AUTHOR BIO:

Steve Muench is consulting product manager on the Business Components for Java Development Team at Oracle and their lead technical evangelist for XML. In his 13 years at Oracle, he's been involved in the support, development, and evangelism of Oracle's application development tools and database and a driving force in helping Oracle development teams from the database server, to application server, to tools, to packaged applications weave XML sensibly into their future development plans and to adopt a component-based development architecture for the future. Steve is the author of Building Oracle XML Applications *(O'Reilly).*

STEVE.MUENCH@ORACLE.COM

Since virtually all valuable Web services both access corporate data and enforce some form of business logic, any productivity gains you can realize during development for these common tasks will prove invaluable as well. This article explores two techniques for database-driven Web service development that I used on a recent project, a system to manage development-team responses to customer questions on the Oracle Technology Network (http://otn.oracle.com) online discussion forum. On this project, due to time pressure, wherever application functionality had already been written by other developers in a reusable way, I did my best to utilize it. In fact, I made quick work of some of my tasks by directly exposing their existing database stored procedures as Web services. For new functionality, I exploited a widely-used J2EE application framework to simplify my data access, XML message handling, and business logic enforcement instead of coding the "design patterns" to implement these aspects by hand. We will examine simplified versions of a few representative cases that I encountered in my application. This should make it easy to understand the basics of the approach I used.

Rolling Up Our Sleeves: Toolset or Tool Belt?

While you're targeting the J2EE platform with your enterprise Web service development, it's clear that you will exercise a healthy mix of J2EE technologies along the way. For example, at a minimum you'll use JDBC APIs for data access, JAXP for XML processing, and JMS for working with message queues, and there's a good chance you'll incorporate some use of EJB for building transactional services that coordinate with other components and resources. I've found that J2EE developers building Web services typically fall into two camps: they either adopt an integrated development environment supporting the entire J2EE development life cycle, or they stitch together a patchwork quilt of favorite command-line tools and toolkits using popular build automation tools such as Apache Ant.

Both are valid approaches, to be sure, but the convenience offered by tools with everything under one roof is getting harder to ignore. Over the last year, the adoption rate of what industry analysts call the "Enterprise Development Studios" for J2EE development from companies like IBM, Borland, and Oracle has grown substantially. Since I'm not much of a command-line handyman, I used Oracle9*i* JDeveloper for my Web services project. Other IDEs may offer features similar to the ones I describe below, but I'll detail what I've done with Oracle9*i* JDeveloper for my project as I'm most familiar with those capabilities.

Repurposing Existing Assets for Fun and Profit

In the online discussion forum management system I work on, we wanted to expose the ability for external customers to program-

ATTN: Developers

STEP UP to the mike and be... HEA

Go to http://developer.sys-con.com

Calling Sleek Geeks Everywhere!

Make sure you have your finger on the pulse of i-Technology...bookmark http://developer.sys-con.com today.

i-Technology News

i-Technology Views

i-Technology Comment

i-Technology Debate

SYS-CON MEDIA

matically check on the discussion forum threads they've posted. After a little initial research, I found that other Oracle developers had already written something that would do most of the job. Their Forum_Info package of database stored procedures had a Forum_Thread_Summary_Info function that accepts the ID number of a forum, and the ID number of a thread in that forum, and returns key summary information about the current state of the discussion thread. A simplified signature of their PL/SQL package looked like this:

```
/* Specification (Public Interface) for
Forum Info Package */
PACKAGE Forum_Info IS
  FUNCTION Forum_Thread_Summary_Info(
the_forum_id  NUMBER,
the_thread_id NUMBER)
  RETURN Forum_Thread;
END;
```

These developers had leveraged the object features of the Oracle9*i* Database to create an object type named Forum_Thread to represent a single, named structure of information about a discussion forum thread. They created its definition using syntax like this:

```
CREATE TYPE Forum_Thread AS OBJECT (
  forum_id              NUMBER(8),
  thread_id             NUMBER(8),
  title                 VARCHAR2(80),
  author                VARCHAR2(40),
  last_reply            DATE,
  awaiting_oracle_reply VARCHAR2(1),
  last_reply_from       VARCHAR2(40),
  total_replies         NUMBER(4)
);
```

Out of curiosity, I asked them to mail me the code for the private implementation of their function so I could have a look, just to make sure their code was using the same interpretation of "awaiting Oracle reply" that I needed to support. Their code used a SELECT statement to retrieve the current state of the forum thread (see Listing 1; code for this article can be found online at www.sys-con.com/webservices/sourcec.cfm).

The query does an outer-join between the Forum_Threads table and the Oracle_Team table so that they can detect whether the last person to reply to the discussion thread was an Oracle team member or a customer, and decode that fact into a letter *Y* or *N*, indicating whether the thread is still waiting on an Oracle team reply. Seeing this confirmed that it was just what I needed.

After creating a named connection to access their database, I used Oracle9*i* JDeveloper's PL/SQL Web service wizard to expose the existing package as a Web service. This involved selecting the appropriate package name from a list and ticking the box next to the particular procedures and functions I wanted to be part of the external Web service interface. After finishing the wizard, the WSDL describing the interface for my Web service had been created for me automatically, and a deployment profile was set up so that deployment to my application server – in my case Oracle9*i* Application Server – took literally just another click.

Testing the New Web Service from a .NET Client

Since a key benefit of Web services is interoperability, I thought it would be a good idea to test my new J2EE Web service from a Microsoft .NET client tool. This would guarantee that customers wanting to programmatically check the status of their forum threads could really do it from any type of client. I pointed my browser to the correct service URL on the server hosting my new **ForumInfo** Web service and was greeted by a browser test page. After exercising the basic functionality from the browser, I copied the URL for the service description and pasted it into the Visual Studio .NET "Add Web Reference…" dialog. As it showed me the ForumInfo service's WSDL contract, I noticed something I hadn't paid attention to before: Oracle9*i* JDeveloper had automatically generated the XML Schema type description shown in Listing 2 for the ForumThread return type into the WSDL.

Visual Studio knew how to leverage this schema information, so after finishing the wizard I had two new C# language objects in my "Web References" folder: a ForumInfo Web service stub, and a ForumThread class that mimicked the structure of my service's ForumThread return type. I cobbled together a simple C# program to verify that everything was working (see Listing 3). Running the program produced the expected output:

```
Title: BC4J deployment questions
Author: Peter Verhage
Waiting on Oracle?: Y
```

So for one aspect of my Web services application, I was already done. If your application server works similarly to Oracle9*i* Application Server, your database-driven Web services will exploit the J2EE container's connection pooling facilities for scalability, and the PL/SQL stored procedure implementation leverages the robustness of the database – and saved me from reinventing the wheel.

Tackling the New Functionality

When I ran out of Web service functionality that I could implement "for free" by exposing existing stored procedures, it was time to design the services that I needed to implement from scratch. One of them was implementing a customer request to supply the complete details of a discussion forum thread as an XML document.

To support other new business functionality in the system, I had already used a combination of tools to design a set of Java business components to cleanly encapsulate the generic business logic and persistence implementation. Rather than coding these business domain classes by hand, I used the built-in J2EE framework called Business Components for Java (BC4J). It's the same one that is used internally by over 800 developers in the Oracle E-Business Suite division, who use it to save time building Oracle's own self-service Web application suite and Web services.

By using various wizards, editors, and UML modeling tools to derive my components from a handy business logic and data access framework, I avoided the pain of having to invent a custom J2EE application architecture on my own. In addition, I spared myself the chore of writing code by hand to implement the scores of J2EE design patterns that my analysis had warned I'd need. As even my simple example confirms, following a J2EE framework approach I was able to focus on writing business code that was relevant to satisfying my end-user functionality requirements instead of having to write low-level application plumbing code.

In addition to a set of "Entity Object" framework components comprising my shared business domain layer, I was able to leverage other BC4J components as well. Both for my Struts-based JSP Web user interface and for my new getForumThread AsXML() Web service, I created query components called "View Objects." These conveniently handle all of the JDBC interaction for me and cleanly encapsulate all the database queries my application has to perform. I used the built-in "Explain Plan" button on the component editor to validate ahead of time that the queries would use all of the

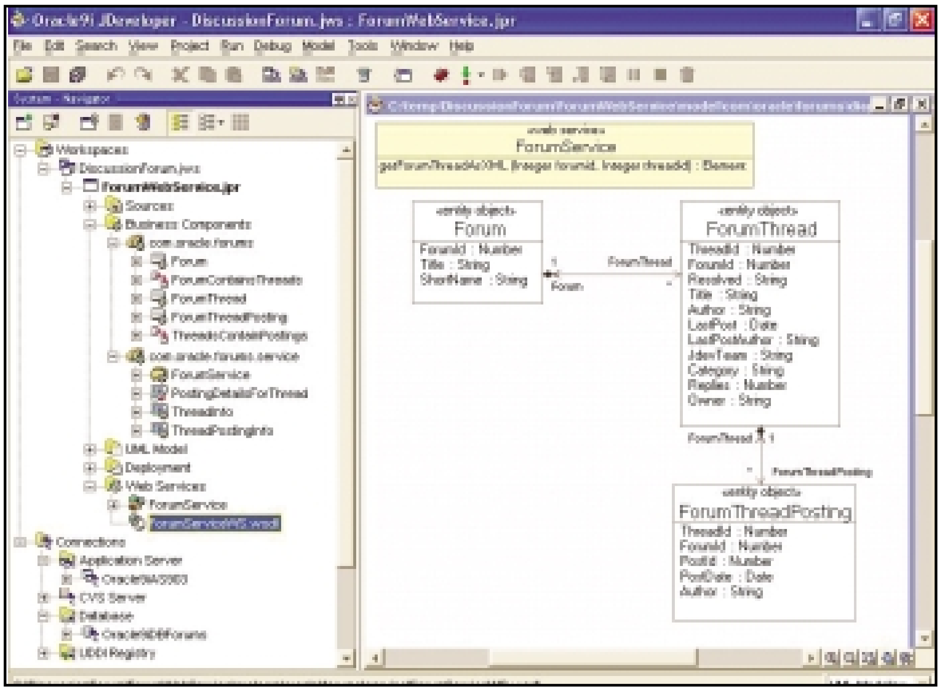


FIGURE 1 **ForumWebService project in Oracle9*i* JDeveloper**

expected indices for best runtime performance. When creation, removal, or modification of rows of business data is required, the view object components delegate automatically to the underlying entity object components that enforce relevant business logic. To complete the job, I used BC4J "Application Module" components to encapsulate any service-related code in a convenient place and centralize the definition of the collections of business data needed by my client applications and Web services.

Figure 1 shows my ForumWebService project in Oracle9*i* JDeveloper, with the ForumService Web service in the UML modeler, along with the three key business domain components of interest to this service. To keep the diagram simple, I've hidden the methods on my domain components, though of course with a click I could show them again for my reference. In the System Navigator, you can see the com.oracle. forums package of shared business domain components (entity objects), and the com.oracle.forums.service package containing my data access components (view objects) and ForumService service component (application module).

The ThreadInfo component (SQL icon) contains the query to retrieve the information I want to expose to clients about forum threads, while the ThreadPosting component has the query for the "slice" of details I want to expose for the individual postings in a discussion thread. By creating the PostingDetailsFor Thread view link component, I define a master/detail data relationship between the two view objects so that each value object "row" produced by the query result of the ThreadInfo view will automatically have a coordinated detail collection of value objects for the postings in that thread.

Since I was leveraging a J2EE framework, my code was limited to a few lines in two methods. First I wrote the following method in my ThreadInfo view object component to encapsulate the setting of its WHERE clause bind variables:

```
/**
  * Encapsulate bind variable setting
  * inside ForumThread view object
  */
 public void findByThreadId(int
forumId,int threadId) {
    setWhereClauseParam(0,new
Integer(forumId));  // set the
    setWhereClauseParam(1,new
Integer(threadId)); // bind vars
    executeQuery();
 }
```

Then I wrote the main getForumThread AsXML() service method in the ForumService application module class:

```
/**
* Use an instance of the "ThreadInfo"
* view object to query the thread data,
* the related thread posting data (via
* the view link), and return a
* multi-level XML message to the client.
* /
  public Element getForumThreadAsXML( int
forumid, int threadid) {
    ThreadInfo info =
(ThreadInfo)getThreadInfo();
    info.findByThreadId(forumid,threa-
did);
    return (Element)
info.writeXML(1,info.XML_OPT_ALL_ROWS);
}
```

Then I just published the application module component as a Web service and deployed it to the target application server – in my case Oracle9*i* Application Server. To write out multilevel hierarchical XML messages, we use just one method, write XML(). It's interesting to note as well that for other aspects of my application's functionality, instead of using writeXML() I found the view object's companion read XML() invaluable. It let me process incoming, multilevel XML messages using my shared business domain components with a single method call.

Conclusion

If you take home one message from this article, I hope it is: "Above all, don't reinvent the wheel!" By using appropriate tools, it can be easy to repurpose existing assets like database stored procedures into Internet-ready Web services that interoperate with any kind of client.

For green field Web services development on the J2EE platform, rather than starting from scratch or risk making costly mistakes on our own, we saw that a J2EE framework tightly integrated with your Web services development tool can provide a proven application architecture out of the box, leaving only the "fun" code to write ourselves.

As I found in my project, you'll likely end up using a combination of these time-saving techniques. Along the way, if you're anything like me, you'll thank yourself time and again for adopting your favorite full life cycle J2EE tool and a J2EE framework that let you finish faster and go home a little earlier every night! ©

Discover Web Services For Yourself

Architect a dynamic discovery with a publicly available service

The Web services phenomenon has finally arrived. All of the developer conferences and symposiums are talking about how Web services will revolutionize the world. Well, it takes action to start a riot, and this article should set your wheels in motion. In this article, you'll learn how to locate available Web services, and systematically integrate discovered services into your existing application by using XPath to parse the services, WSDL file.

AUTHOR BIO:

Ron West is a senior applications developer with PaperThin, Inc., a privately held Web content management vendor headquartered in Quincy, Massachusetts. Ron has been working with Web applications for seven years, is one of the directors of the Rhode Island ColdFusion User Group, and is an established writer for several industry publications.

RWEST@PAPERTHIN.COM

First, we need a service to work with. For our example we'll focus on the RPC-style Web services as these are the most common and easiest of the Web service solutions to implement. Since UDDI is still a relatively distant option for discovering services, we will turn to our good friends over at Xmethods.net, who have an extensive listing of available Web services. Let's select the Global Ski Resort Finder (GSRF) service (a nifty little tool that locates ski resorts within a certain radius of a given longitude/latitude point) and take a brief look at the WSDL file, located at www.transactionalweb.com/SOAP/globalskilocator.wsdl.

WSDL Architecture

The WSDL file, which is simply an XML file that describes the Web service, contains the following general format (for brevity, we will discuss the main contents of a WSDL file and try to apply them to our example file).

<description> element has at least one each of the following children elements: <type/>, <message/>, <portType/>, <service Type/>, <binding/>, <service/>, </description>.

The <description> element is the root element. This is where you see the namespaces for the collective properties of the WSDL file. The <type> element contains the definitions of the datatypes used for the services operations. In the GSRF file, notice that the <type> element contains a definition for the "Entry" datatype, which in this case is of type array. (We won't discuss this custom type in this article.) Next is the <message> element that defines the structure of the messages sent between the client and the host. We have two <message> elements of names "findWithin RadiusResponse" and "ski". Notice that each <message> element contains a <part> element. The <part> element defines the encoding type for that particular message (either a response message or a call message). After that, we have the <portType> element that defines the parameters for all operations available. The <portType> attribute "name" is used as an internal reference, which we will tie together a little later. Next comes the <binding> element, which is used to provide the operation details.

In our example, we see an <operation> element inside the <binding> element, which defines the input and output parameters for the service named "findWithinRadius". Finally, we have our <service> element, which contains the names of the services available. The GSRF has one <operation> tag with a name attribute whose value is "findWithinRadius". The <operation> tag will have any of the following children elements: <input>, <output>, and/or <fault>. The children tags inform us of the existence of input, output, and fault parameters.

XPath and Scripting

XPath has been a W3C recommendation since November 1999. XPath is designed to provide a common interface to searching XML documents for XSLT and Xpointer, but it can also be incorporated into scripting languages such as JSP, PHP, and ColdFusion. By implementing XPath syntax and expressions in a scripting language, you can search the WSDL file and dynamically map all the properties of a Web service. Most scripting languages that have integrated XPath support will allow XML path searches. Specifically, it's possible to locate nodes and discover XML attributes of those nodes. Within the XPath recommendation, there are additional node tests available that can be used to discover further attributes of the returned XML nodes. We will assume here that the scripting language makes these attributes available with a standard API. In our example, we would take the following steps to discover the Web service properties:

1. Make sure that we are working with an RPC-style service, using this bit of code:

```
//*[contains(name(),'binding')][@style=]
```

Select any nodes named 'binding' that have an attribute named 'style' with the value 'encoded'

2. Next, we want to discover the name of the service:

```
//*[contains(name(), 'service')]
```

Select the <service> node. (This will return the name of the Web service.)

3. Now, let's discover what ports (methods) are available for this service:

```
//*[contains(name(), 'binding')]/[@name]
```

Select all of the children of the <binding> node that have children with an attribute name.

4. From step 3, we now know that the context node (our first node returned from the search) will have an attribute 'name' and the value of this attribute will be the method. (It is possible to have multiple methods returned from this initial query, which means each sibling node, at the same level as the context node, would also have an attribute 'name' and the value of it would be its method name.)

5. Next, in order to discover the input, output, and/or fault codes available for each port (method), we make the following call on our WSDL file:

```
//*[contains(name(), 'binding')]/*[con
   tains(name(),
'operation')][@name='$port']
```

Select all of the nodes that exist in the <binding> node that have the name 'operation' and contain the attribute 'name' whose value is $port' (note: the use of the $ here denotes that we are working with a variable of some sort). Basically you would want to do this for each <operation> node returned from step 2.

6. From step 4, we can search for the existence of either an <input> and/or <output> and/or <fault> node, which indicates what is in use for this service. Commonly, a service will have an <input> and <output>, but it's entirely possible for a service to have just an <input> or just an <output>.

<breath> What we have done so far is significant enough for reflection. Basically, we have just discovered the Method names for all of the services available and for each Method, whether or not it has input, output, and/or fault messages. This is only the beginning, but it is a solid foundation with which to work.

```
<breath/>
```

7. The portType will help us discover the mapping for the datatypes of the messages related to the service. For each found node in step 5, execute the following to discover the mapping for the node's datatype:

```
//*[contains(name(), 'portType')][*[con
     tains(name(),
'operation')][@name='$port']]/*/*
```

Select all nodes whose names contain 'portType' with children nodes whose names contain 'operation' and have an attributed called 'name' with value $port. (Note: XML is a case-sensitive language so "findwithinradius" is NOT the same as "findWithinRadius".)

8. What we need from step 7 is the value of the message 'attribute' for each of the available messages (input/output and/or fault). The value for each will help us find in the WSDL document the datatype expected from the client for that message. (Note: each message will be prefaced with the namespace for its encoding style. Commonly, this will appear as tns:mapping. In our case, we are merely interested in the 'mapping', not the 'tns:'.)

9. For each value returned by step 7, we can perform the following search to determine the location of the datatype:

```
//*[contains(name(),
'message'][@name=$mapping]
```

Select all nodes whose names contain 'message' with an attribute 'name' whose value is $mapping (again, we use the $ to denote dynamic variable here)

10. The data from step 8 will contain a <part> element(s) that define the name of the expected part, but more importantly, we now have the datatype. Here is where things get a little tricky. The datatype can either be a standard datatype like String or Boolean, in which case we are done with the discovery process. However, if this datatype is a custom or complex datatype (denoted by a prefix other than "tns:"), a little more discovery is required. The complex data type is just that - complex. For now, we will leave that for another article.

What Should We See?

From our example (The Global Ski Resort Finder), you should have discovered the following properties:

1. A service named "QueryInterfaceService".
2. A port (method) named "findWithin Radius" with an input and output parameter.
3. The input message for the "findWithin Radius" port was mapped to the "intf:ski" message type.
4. The output message for the "findWithin Radius" port was mapped to the "intf:find WithinRadiusResponse" message type.
5. The "ski" message contained three parts
   a. Latitude (of type string)
   b. Longitude (of type string)
   c. Radius (of type string)
6. The "findWithinRadiusResponse" message contained one part: Noname (of type ArrayOfEntry – the locally defined complex type).

You now have all of the information you need to dynamically call this service and systematically incorporate it into your application.

The Revolution Begins

We all know that Web services are here, and that Web services create a new paradigm for the distributed application model. Once the security model is put in place (hopefully by the WS-I), we can use examples like these to at least understand what is needed to architect dynamic discovery and use of the ever-growing population of publicly available services. Happy Web servicing!

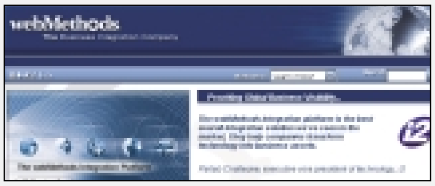
webMethods, J.D. Edwards, and HP Collaborate on BPM Solution

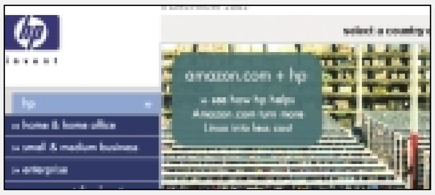
(Fairfax, VA and Denver, CO) – webMethods, Inc., and J.D. Edwards have announced an initiative between webMethods and HP to add significant value to J.D. Edwards' XPI offering. webMethods and HP will provide systems management technology for J.D. Edwards' applications, making it easier to monitor and manage the business processes enabled by J.D. Edwards' XPI technology.

J.D. Edwards currently embeds webMethods' integration software into their collaborative enterprise applications, including J.D. Edwards 5, creating an integration backbone for J.D. Edwards' technology, J.D. Edwards XPI, which decreases the cost of integrating applications through a centralized model of integration.

webMethods and HP collaborated to write the Open Management Interface specification, linking the technology and business components that drive an organization, exposing them in a common interface for monitoring and analysis through the HP OpenView Smart Plug-in for webMethods.

www.webMethods.com, www.hp.com, www.jdedwards.com





OASIS Forms e-Gov Technical Committee

(Boston) – The OASIS interoperability consortium has announced that it is providing an international forum for governments to voice their needs and requirements with respect to XML-based standards.

The OASIS e-Gov Technical Committee will assist in the electronic delivery of services to citizens and businesses through the coordination and adoption of XML standards.

The committee will coordinate input from governments on emerging technologies, such as ebXML and Web services, to ensure that existing specifications are not developed solely for the benefit of the private sector.

www.oasis-open.org

AdventNet Debuts ManageEngine Suite

(Pleasanton, CA) – AdventNet, a provider of standards-based J2EE management solutions, has introduced ManageEngine Suite 5, the first software framework that allows customers using the Java Management Extensions (JMX) standard to track the business impact of application performance. ManageEngine Suite 5 automates instrumentation for the management of business applications, provides automated tools to build custom consoles, and integrates with existing management consoles. Developers can graphically pick attributes to monitor and measure, creating rules for when IT or even business managers are alerted to kinks in the system. No additional code needs to be written.

www.adventnet.com

Fiorano Software Ships Tifosi 2002

(Los Gatos, CA) – Fiorano Software, Inc., has released Tifosi 2002, a service-based integration platform to extend its offerings in the EAI, BPM, and B2B integration market. Tifosi is based on super-peer architecture that facilitates the composition, deployment, and management of distributed enterprise applications. It also links individual enterprises together for extended process efficiency across the supply chain, allowing them more flexibility and adaptability to rapidly changing business requirements.

www.fiorano.com

DataDirect Technologies Adds Distributed Transaction Support to DataDirect Connect for .NET

(Rockville, MD) – DataDirect Technologies, Inc., has announced DataDirect Connect for .NET 1.1, newly updated data connectivity components that enable software developers to access enterprise data using .NET. The new release adds support for distributed transactions for Oracle and Sybase, and updated support for several Oracle data types.

DataDirect Connect for .NET allows developers to implement serviced components that require distributed transaction support and use ADO.NET Data Providers.

The MS DTC can update multiple databases and files from a single application, update geographically distributed databases, and update databases that have been partitioned for scalability.

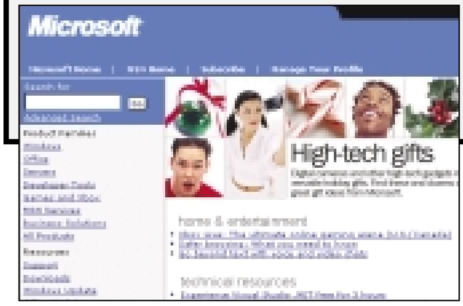
www.datadirect-technologies.com

Microsoft Launches Small Business Manager 7.0

(Fargo, ND) – Microsoft Business Solutions Small Business Manager 7.0, an affordable, integrated software application developed to help small businesses manage a broad range of business functions, is now available.

The new offering enables small businesses to increase productivity and realize maximum flexibility through streamlined sales and purchasing, tight fiscal and inventory control, and enhanced integration. Built for small businesses that need more than basic bookkeeping software, Microsoft Business Solutions Small Business Manager 7.0 includes includes integrated modules that allow small businesses to more efficiently manage financial, banking, inventory, sales, purchasing, U.S. payroll, and reporting and analysis functions.

www.microsoft.com



Fujitsu Selects The Mind Electric to Develop Open Grid Services Architecture

(Dallas) – The Mind Electric (TME), a provider of software for service-oriented architectures, will collaborate with Fujitsu Laboratories to develop innovative applications of grid technologies based on service-oriented architectures and Web services.

Fujitsu Laboratories Europe and TME are collaborating to develop OGSA extensions to GLUE to support a cross-platform grid infrastructure that is secure and reliable.

www.themindelectric.com, www.fujitsu.com

Continued from page 4

vas can be very handy for creating the XML-to-HTML transformation.

The instant gratification using the IDE-style metaphor of "edit/compile/run" that most of us have grown accustomed to should apply to writing transformations as well. There's no substitute for being able to automatically apply a stylesheet to a piece of data and immediately see the output. For some folks the IDE metaphor is more often "edit/compile/debug," and that should be OK in this world too. Think of being able to set breakpoints in the XSLT document, or in an associated Java extension, and being able to step through a transformation and watch it happen. How about back-mapping from a destination document – being able to click on the piece of data in the output document and to automatically be brought to the line of XSLT code that produced that data? You just don't get that in Notepad.

As you and your IT staff gain responsibility for a greater share of integration projects, XML technology, skills, and tool sets will become increasingly important to the success of your integration projects, and indeed your success as well.

WSJ ADVERTISER INDEX

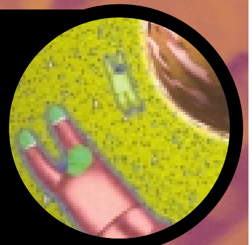
ADVERTISER	URL	PHONE	PAGE
Altova	www.altova.com		13
BEA Systems	dev2dev.bea.com/useworkshop		60
BEA eWorld	www.bea-eworld.com		20
Black Hat	www.blackhat.com		17
JDJ Store	www.jdjstore.com	800-303-JAVA	39
LinuxWorld Conference & Expo	www.linuxworldexpo.com		26, 27
Parasoft	www.parasoft.com/wsj1	888-305-0041	5
Sitraka (now part of Quest Software)	www.sitraka.com/jclass/ws	800-663-4723	9
Sitraka (now part of Quest Software)	www.sitraka.com/jclass/ws	800-663-4723	59
SpiritSoft	www.spiritsoft.com/climber	508-473-3227	6
Sonic Software	www.sonicsoftware.com/websj		2
SYS-CON Media	www.sys-con.com	888-303-5282	49
SYS-CON Media	www.sys-con.com/suboffer.cfm	888-303-5282	44, 45
SYS-CON Media	http://developer.sys-con.com	888-303-5282	51
Web Services Edge 2003	www.sys-con.com	201-802-3069	31
Web Services Journal	www.wsj2.com	888-303-5282	35
Wireless Developer's Journal	www.sys-con.com	201-802-3020	25

IN THE NEXT ISSUE OF WSJ...

Focus: Managing Web Services

Web Service Versioning and Deprecation

A system that's lightweight, flexible, and for corporate use requires minimal development effort using one or more internal UDDI registries

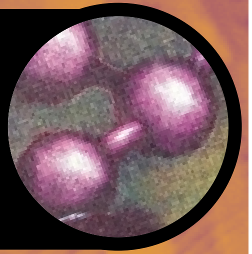


Web Service Management

The benefits of open interoperability only increase the complexity of the computing environment that has to be managed

A Publish/Subscribe Mechanism for Web Services

Develop this kind of service by extending the functionality of an existing framework



Web Services: Reaching Back to Unlock Legacy Systems

Today, companies can get more from their mainframe legacy systems by integrating the data with a Web services solution

WebServices .NET J2EE XML JOURNAL

Beyond the Trinity

Patrick Gannon

Patrick Gannon is president and CEO of OASIS, the not-for-profit, global consortium that drives the development, convergence, and adoption of e-business standards.

Thankfully, the myth that Web services standards begin and end with SOAP, WSDL, and UDDI is fast dissolving. Most developers now appreciate that mission-critical Web services must also extend to include standards in security, resource management, remote application access, choreography, semantics, and much more.

A number of consortia and industry groups are defining the expanding scope of Web services standards. The two major standards-setting bodies for Web services are the World Wide Web Consortium (W3C), and the Organization for the Advancement of Structured Information Standards (OASIS). (WS-I, the Web Services Interoperability Organization, does not develop standards; its mission is to provide guidance, best practices, and resources for developing solutions. WS-I's deliverables [profile definitions, testing tools and sample applications] are based on the standards developed by OASIS and W3C.)

OASIS and W3C maintain close ties with one another. Sharing many of the same members, the groups acknowledge that Web services standards are far too broad for any one organization to own, and they work hard to avoid overlap or duplication of effort. Nevertheless, their approaches are substantially different.

The OASIS technical process lets members take the lead, identifying needs based on their own technology and domain expertise and forming technical committees to address those needs. The membership is a rich mixture of software vendors, end-user companies, governmental agencies, industry associations, educational institutions, and individuals.

There are two levels of approved work at OASIS. The first, OASIS Committee Specifications, refers to work that is approved by members of an OASIS Technical Committee. The second, OASIS Open Standards, signifies work that has been approved by the committee members, proven in at least three real-world implementations, passed a 30-day public review, and ratified by the OASIS membership-at-large.

The Security Assertion Markup Language (SAML) has achieved OASIS Open Standard status as an XML-based framework for Web services that allows the exchange of authentication and authorization information among business partners. SAML enables Web-based security interoperability functions, such as single sign-on, across sites hosted by multiple companies. It is an important industry standard for federating diverse security domains across Web services environments and incorporates industry-standard protocols and messaging frameworks from W3C, such as XML Signature, XML Encryption, and SOAP. Most vendors of Web access management solutions have committed to SAML and are implementing the specification in their products.

The WS-Security specification provides a foundation for secure Web services, laying the groundwork for higher-level facilities such as federation, policy, and trust. WS-Security defines a standard set of SOAP extensions, or message headers, that can be used to implement integrity and confidentiality in applications. It is one of the first Web services standards to support, integrate, and unify multiple security models, mechanisms, and technologies, allowing a variety of systems to interoperate in a platform- and language-neutral manner.

The widespread need for the integration of systems and network management tools is causing the industry to take a more holistic approach to the management of networks – and Web services provide the ideal vehicle for making that happen. OASIS is developing a protocol that will enable businesses to manage their own Web services and oversee their interaction with services offered by other companies. The OASIS Management Protocol is being designed to manage desktops, services, and networks across an enterprise or Internet environment, allowing companies to manage systems regardless of the platform they use.

Web Services for Interactive Applications (WSIA) provides interactive application access through a coordinated set of XML vocabularies and Web services interfaces that allow companies to deliver Web applications to end users through a variety of channels – directly to a browser, indirectly through a portal, or embedded into a third-party Web application. It will enable any Web application – a package tracker, a calendar application, a stock quote – to be delivered and displayed to an end user as a Web service, regardless of the underlying Web platform, vendor-specific application format, or display device. With WSIA, companies will be free to syndicate their applications across different portals and Web site platforms without being limited by proprietary products. They will be able to dynamically share Web services without the time and labor of creating multiple vendor-specific connectors written to different Web languages.

Many continue to define Web services against a checklist of standards that is much too short. In reality, more than 50 different specifications are under development within OASIS alone, and most of these relate to Web services to some degree.

From foundational standards such as UDDI, to intermediary specifications such as Web Services for Remote Portals (WSRP) and Business Transaction Protocol (BTP), from traditional technologies such as Public Key Infrastructure (PKI), to broad solutions such as ebXML, OASIS standards go beyond the bare-bones definition proposed by many tunnel visionaries. Indeed, mission-critical Web services standards require interoperability on a much grander scale.

Sitraka

(now part of Quest Software)

www.sitraka.com/jclass/ws

BEA Systems

dev2dev.bea.com/useworksho