SYS-CON

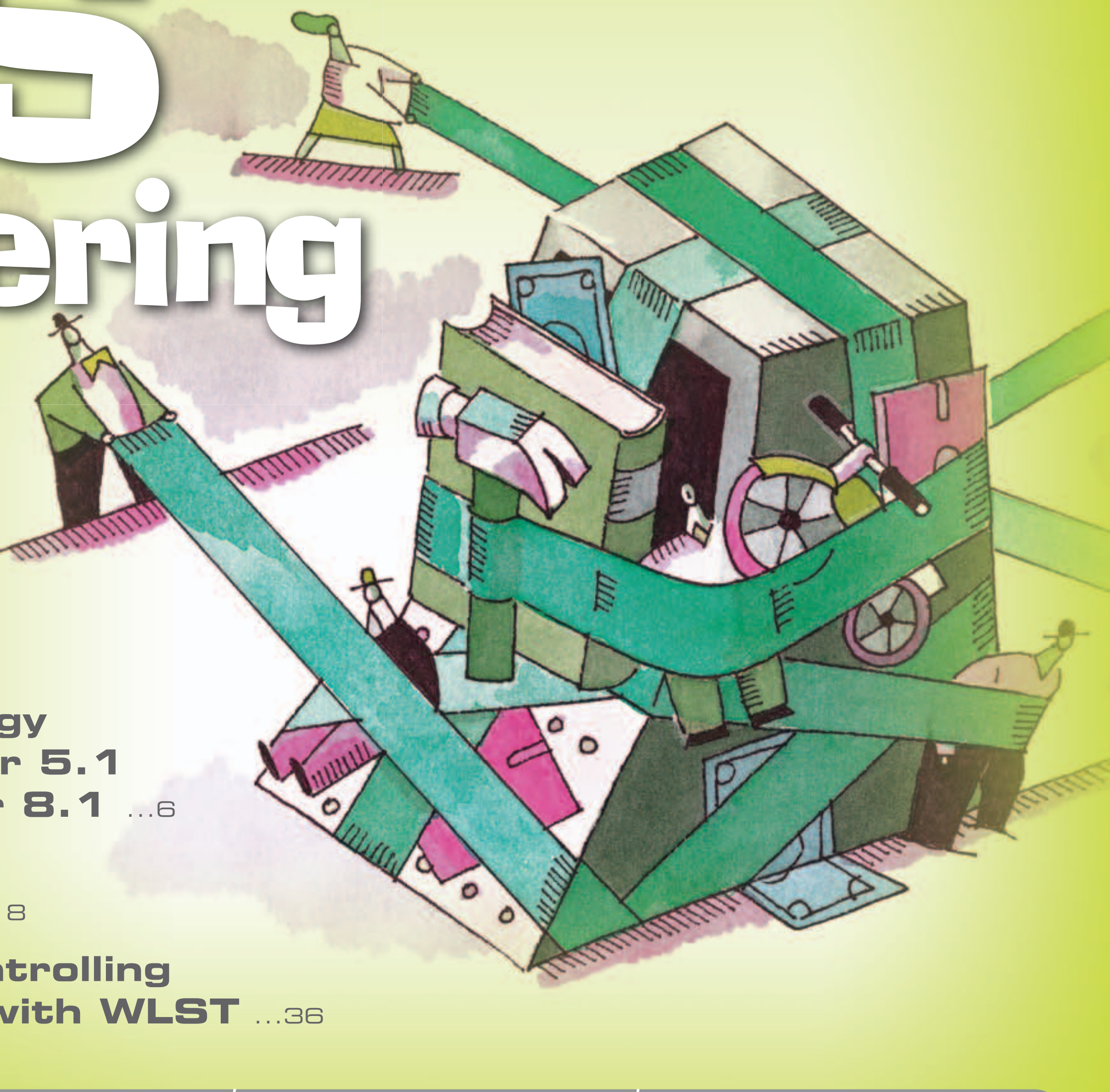
A LOOK AHEAD TO THE SERVICE-ORIENTED WORLD PG 12

MAR/APR 2005 VOL:4 ISSUE:2

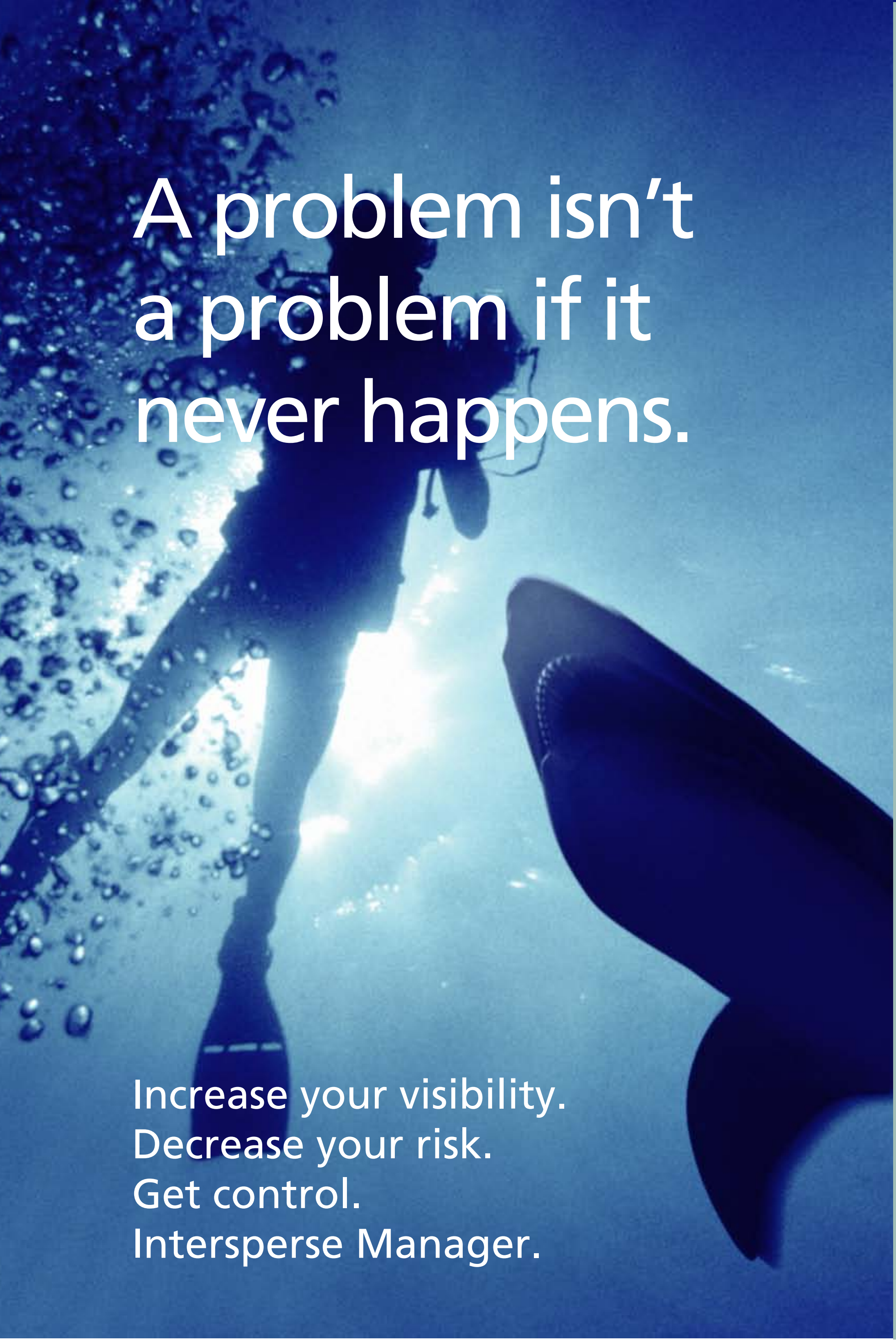
wldj™

THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC™ PROFESSIONALS

WWW.WLDJ.COM



$8.99US $9.99CAN


A problem isn't a problem if it never happens.
Increase your visibility.
Decrease your risk.
Get control.
Intersperse Manager.

Enterprise information systems are more complex – and vital – than ever. But visibility into these dynamic architectures is minimal, increasing the chances of system failure and serious risk to your company.

Intersperse Manager™ removes the risks by proactively monitoring and managing distributed and SOA applications deployed on BEA. It gives users the ability to dive deeply into system infrastructure. To quickly pinpoint and correlate the root causes of problems. To automatically or manually correct issues in real time.

Intersperse Manager gives you complete visibility – and that puts you in complete control.

Call Intersperse today for a free demonstration

866.499.2202

intersperse™

Management solutions for the evolving enterprise.

www.intersperse.com

Businesses spend more than $80 billion annually on integration.

$19 billion of that cost is spent on manually reconciling data between systems.

Automating the Exchange of Data Across an SOA

PANTERO

Over 40% of IT development time is spent on manually reconciling and validating the data shared between systems - now there is a better way.

Pantero's unique Exchange Modeling™ technology captures data reconciliation and validation operations as metadata and generates Shared Data Services where operations can be enforced, reused, and dynamically redeployed without incurring additional integration costs. And, Pantero is seamlessly integrated into the BEA 8.1 Platform.

Contact us today!

781-890-2890

www.pantero.com

Midnight Madness

By Joe Mitchko

As the father of an avid teenage video game enthusiast, I was a bit amused late last year by all of the excitement and anticipation surrounding the upcoming release of Halo 2.

For months leading up to the November 9 release date, I heard all the buzz from my oldest son about how great it's going to be, how much better the graphics will be over the current game, and so on. Not really being much of a gaming enthusiast myself, I let it go in one ear and out the other.

As the fateful day approached, my son was eager to pick up his reserved copy at the local video game store. The whole gaming world seemed transfixed on its midnight release, so it really didn't surprise me when he asked if he could camp out at the store till 12 with his friends, and of course, spend the rest of the night and the next day in virtual ecstasy. Of course, being a good father, the answer was no, and I explained to him what truant officers do for a living. Anyway, the day came and went and somebody got a lot richer. Sales exceeded a blockbuster movie. And of course, my son was in seventh heaven.

Now, flip the calendar forward a little more than a month, and another highly anticipated release is ready to roll out on to the computing stage. This one slightly different – devilishly so I suppose. With all the buzz coming from BEA over the last year about Service Oriented Architecture (SOA), and with hints here and there as to what improvements WebLogic 9.0 (code named Diablo) will have over its predecessor (version 8.1), I was eager and ready to download the beta version. What new features would I behold? What new levels of Web Service integration would I be able to implement? If I could only get a peek at some of the release documents and details about what they were up to. This one was under tight wraps, for sure. With a few days to go before to the beta's December 16 release date, expectations were high. And, it even crossed my mind that maybe; just maybe, I should stay up until midnight to be one of the first to download a copy.

After realizing that BEA is on the West Coast, and that that meant 3am Eastern time, I dropped that idea. Besides, I had work the next day.

Anyway, I eventually did get to download a copy a few days after the release (too many meetings at work), and was pleasantly surprised with all the improvements the 9.0 version has over its predecessor. First, a few enhancements in management and administration will make it easier to deploy and manage applications. The new portal-driven management console makes it easier than ever to deploy and manage applications and services. Improvements in logging and monitoring bring us a step closer to the SOA world envisioned by the industry and espoused by BEA. The improvements and new features are really too numerous to mention here, but after looking through BEA's 100-page release note, I think things are moving forward in a positive direction. It's apparent that BEA is taking on the Enterprise Service Bus (ESB) competition with the 9.0 release and its clustered JMS service implementation.

Strangely absent from the beta are any documented improvements to the Workshop IDE and underlying Beehive-based framework. Is the fate of Workshop sealed, and will we eventually see Eclipse/Pollinate take over once it's reached critical mass? If you haven't downloaded the 9.0 beta yet, I would suggest you do so. It's worth staying up for.

Author Bio:
Joe Mitchko is the editor-in-chief of ***WLDJ*** and a senior technical specialist for a leading consulting services company.

Contact: joe@sys-con.com

wldj THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC™ PROFESSIONALS

EDITORIAL ADVISORY BOARD
Lewis Cirne, Wayne Lesley Lund, Chris Peltz, Sean Rhody

EDITORIAL
Founding Editor
Peter Zadrozny
Editor-in-Chief
Joe Mitchko joe@sys-con.com
Editor
Nancy Valentine nancy@sys-con.com
Associate Editor
Seta Papazian seta@sys-on.com
Research Editor
Bahadir Karuv, PhD bahadir@sys-con.com

PRODUCTION
Production Consultant
Jim Morgan jim@sys-con.com
Lead Designer
Abraham Addo abraham@sys-con.com
Art Director
Alex Botero alex@sys-con.com
Associate Art Directors
Louis F. Cuffari louis@sys-con.com
Richard Silverberg richards@sys-con.com
Tami Beatty tami@sys-con.com
Andrea Boden andrea@sys-con.com

CONTRIBUTORS TO THIS ISSUE
Joe Mitchko, Manjunath Ganimasty, Thomas Erl, Vijay Chinta, Thomas Kunnumpurath, Channing Benson, Raffi Basmajian, Ignacio Coloma, G.V.B Subrahmanyam, Blair Toylor

EDITORIAL OFFICES
SYS-CON MEDIA
135 Chestnut Ridge Rd., Montvale, NJ 07645
Telephone: 201 802-3000 Fax: 201 782-9638
WLDJ (ISSN #1535-9581)
is published bimonthly (6 times a year)
by SYS-CON Publications, Inc.

Postmaster: send address changes to:
WLDJ: SYS-CON MEDIA
135 Chestnut Ridge Rd., Montvale, NJ 07645

Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ
FOR LIST RENTAL INFORMATION:
Kevin Collopy: 845 731-2684,
kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832,
frank.cipolla@epostdirect.com

REPRINTS
For promotional reprints, contact reprint coordinator Kristin Kuhnle, kristin@sys-con.com. SYS-CON Publications, Inc., reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

Why is it that some Java guys are more relaxed than others?

These days, with everything from customer service to sales and distribution running on Java, keeping your enterprise applications available can be pretty stressful.

Unless of course, you've discovered the power of Wily. No other software offers our level of insight. Which means you'll be able to monitor transactions and collaborate with colleagues in real-time. Even more important, you'll be able to optimize performance across the entire application—end-to-end.

So put Wily to work. And keep performance problems from pushing you over the edge.

Get Wily.™

wily technology

1 888 GET WILY | wilytech.com

A Migration Strategy for WebLogic Server 5.1 to WebLogic Server 8.1

By Manju Ganimasty

OVERVIEW, ADVANTAGES, AND NEW FEATURES

BEA retired its WebLogic Server 5.1 on February 1, 2004. It had been supported for four years prior to retirement. One of the options for migrating applications that are running on 5.1 is to use WebLogic Server 8.1, which provides a range of new features for J2EE 1.3-based applications.

I'll give you an overview of the significantly important features in WebLogic Server 8.1 and the high-level activities involved in migrating applications from WebLogic Server 5.1 to 8.1.

New Features in WebLogic Server 8.1

JDBC Features

WebLogic Server 8.1 SP1 includes a new JDBC driver from BEA for connecting to a Microsoft SQL Server database. The BEA WebLogic Type 4 JDBC MS SQL Server driver replaces the WebLogic jDriver for Microsoft SQL Server that's been deprecated. The new driver:

- offers JDBC 3.0 compliance and better performance
- supports distributed transactions through XA support
- supports multilingual application development on any operating system platform to access both Unicode- and non-Unicode-enabled databases

Administration Features

WebLogic Server 8.1 provides a ***System Administration Console*** that's a web browser-based graphical user interface (GUI) to manage a WebLogic Server domain. One instance of WebLogic Server in each domain is configured as an **Administration Server**. The Administration Server provides a central point for managing a WebLogic Server domain. All other WebLogic Server instances in a domain are called **Managed servers**.

The features offered by WebLogic Server 8.1 System Administration Console are:

- Configure, start, and stop WebLogic Server instances
- Configure WebLogic Server clusters
- Configure WebLogic Server services such as database connectivity (JDBC) and messaging (JMS)
- Configure security parameters including managing users, groups, and roles
- Configure and deploy applications
- Monitor server and application performance
- View server and domain log files
- View application deployment descriptors
- Edit selected runtime application deployment descriptor elements

The Administration Console provides additional runtime data for servers running with the JRockit Virtual Machine.

Security Features

The SSL implementation of WebLogic Server supports KeyStores for storing Private Keys and trusted Certificate Authorities (CAs). KeyStores add a level of protection to the flat files used in past releases of WebLogic Server.

WebLogic Server 8.1 supports standard J2EE security technologies such as Java Authentication and Authorization (JAAS), Java Secure Sockets

Author Bio:
Manjunath Ganimasty is a solutions architect and lead developer working with HP's Imaging & Printing Group (IPG) Information Technology (IT) team. He has recently been working on projects with the Consumer Direct Integration Platform, helping to develop, deploy, and manage applications in the Web services environment. He has worked on telecommunication and supply chain related projects for Hewlett Packard for the past four years. Manjunath has a Bachelor's degree in Computer Science from Bangalore University and a Masters Degree from Georgia State University.

Contact:
manjunathgm@yahoo.com

PRESIDENT & CEO
Fuat Kircaali fuat@sys-con.com

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

GROUP PUBLISHER
Jeremy Geelan jeremy@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 Manager
Megan Mussa megan@sys-con.com

Advertising Sales Manager
Dennis Leavey dennis@sys-con.com

Associate Sales Managers
Kristin Kuhnle kristin@sys-con.com
Dorothy Gil dorothy@sys-con.com
Kim Hughes kim@sys-con.com

SYS-CON EVENTS

President, Events
Grisha Davida grisha@sys-con.com

National Sales Manager
Jim Hanchrow jimh@sys-con.com

CUSTOMER RELATIONS

Circulation Service Coordinators
Edna Earle Russell edna@sys-con.com
Linda Lipton linda@sys-con.com
Monique Floyd monique@sys-con.com

Manager, JDJ Store
Brunilda Staropoli bruni@sys-con.com

SYS-CON.COM

VP, Information Systems
Robert Diamond robert@sys-con.com

Web Designers
Stephen Kilmurray stephen@sys-con.com
Matthew Pollotta matthew@sys-con.com

Online Editor
Martin Wezdecki martin@sys-con.com

ACCOUNTING

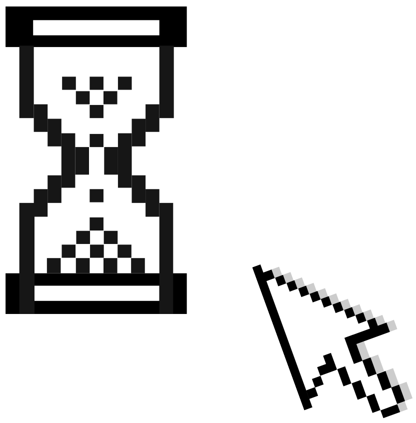
Financial Analyst
Joan LaRose joan@sys-con.com

Accounts Payable
Betty White betty@sys-con.com

Credits & Accounts Receivable
Stephen Michelin smichelin@sys-con.com

SUBSCRIPTIONS

subscribe@sys-con.com
Call 1-888-303-5282
For subscriptions and requests for bulk orders, please send your letters to Subscription Department
Cover Price: $8.99/issue
Domestic: $149/yr (12 issues)
Canada/Mexico: $169/yr All other countries $179/yr
(U.S. Banks or Money Orders)
Back issues: $12 U.S. (plus S&H)



World Wide Wait.

Don’t Leave Your Customers Hanging — Ensure a Positive End User Experience with Quest Software.

They don’t care where the problem is. All that matters is their experience using your site. But can you quickly detect and diagnose a problem and deploy the right expert to fix it?

Quest Software’s solutions for J2EE application performance management shed light on the true end user experience and help you isolate and fix problems across all tiers. Whether a problem lies in your server cluster, database, framework or Java code, nothing gets you to resolution and optimal performance faster.

Don’t just cross your fingers and hope for the best. Get more with Quest.

Download the free white paper,
"Measuring J2EE Application Performance in Production"
at www.quest.com/wldj

Application Management | Database Management | Windows Management

QUEST SOFTWARE®

TABLE 1

Standard	Version
J2EE	1.3
JDKs	1.4
J2EE EJB	2.0 and 1.1
J2EE JMS	1.0.2b
J2EE JDBC (with third-party drivers)	2
MS SQL jDriver	1
Oracle OCI jDriver	1.0 and some 2.0 features (batching)
J2EE JNDI	1.2
OTS/JTA	1.2 and 1.0.1b
J2EE Servlet	2.3 and 2.2
J2EE JSP	1.2 and 1.1
RMI/IIOP	1
JMX	1
JavaMail	1.2
JAAS	1.0 Full
J2EE CA	1
JCE	1.4
Java RMI	1
JAXP	1.1
JAX-RPC	1

Java standards supported in WebLogic Server 8.1

TABLE 2

Standard	Version
SOAP	1.1 and 1.2
WSDL	1.1
UDDI	v1 and v2
WS-Security	1

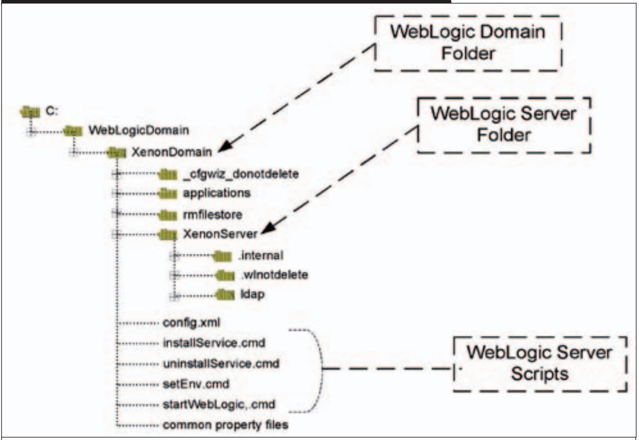
Web Services standards supported in WebLogic Server 8.1

TABLE 3

Standard	Version
SSL	v3
X.509	v3
LDAP	v2
TLS	v1
HTTP	1.1

Other standards supported in WebLogic Server 8.1

FIGURE 1



Example directory structure for a typical WebLogic domain

Extension (JSSE), and Java Cryptography Extensions (JCE).

Supported Standards

WebLogic Server 8.1 supports the Java standards shown in Table 1.

WebLogic Server 8.1 supports the Web Services Standards shown in Table 2.

Table 3 shows some additional standards supported in WebLogic Server 8.1.

Migration Strategy

Steps (Tasks) for Migrating from WebLogic 5.1 to WebLogic 8.1

Creation of a WebLogic Domain

A domain is the basic administration unit for WebLogic Server instances. It consists of one or more WebLogic Server instances that can be managed with a single administration server. A domain can include multiple WebLogic Server clusters or non-clustered WebLogic Server instances. A minimal domain can contain only one WebLogic Server instance, which functions as both an administration server and as a managed server. But, it's best to use a dedicated administration server and create one or more managed servers, depending on the applications.

Each server instance in a WebLogic environment must have a unique name, regardless of the domain or cluster in which it resides, or whether it's an administration server or a managed server. In a domain each server, machine, cluster, virtual host, and any other resource type must be uniquely named and must not use the same name as the domain. For WebLogic JMS, this strict unique naming rule also applies to JMS resources, such as JMS servers and stores in multi-domain environments when using the WebLogic Messaging Bridge or the Foreign JMS Server feature for intra-domain operability.

A simple production environment can consist of a domain with several managed servers that host applications and an administration server to do the management operations. In this configuration, applications and resources are deployed to individual managed servers; similarly, clients that access the application connect to an individual managed server.

Production environments that require increased application performance, throughput, or availability can configure two or more managed servers as a cluster. Clustering allows multiple managed server to operate as a single unit to host applications and resources.

The managed servers in a production WebLogic Server environment are often distributed across multiple machines and geographic locations.

Node Manager is a Java utility that runs as a process separate from WebLogic Server and lets you perform common operations tasks for a managed server, regardless of its location with respect to its administration server. While using Node Manager is optional, it provides valuable benefits if your WebLogic Server environment hosts applications with high-availability requirements.

If you run Node Manager on a machine that hosts managed servers, you can start and stop the managed servers remotely using the Administration Console or command line. Node Manager can also automatically restart a managed server after an unexpected failure.

A Node Manager process isn't associated with a specific WebLogic domain. Node Manager resides outside the scope of a domain, and you can use a single Node Manager process to start managed servers in any WebLogic Server domain that it can access.

WebLogic Server administration domain and server configurations can be created using the ***Configuration Wizard***, which can also be used to configure resources like database connectivity (JDBC), Messaging Services (JMS), security groups, security roles, and user accounts. It can also be used to modify existing domains.

Understanding WebLogic Domain Directory Structure

The servers and applications in WebLogic Server 8.1 are managed in WebLogic Server domains. It's best to locate the domain directories outside the WebLogic Server installation directory. They can be in any location that can access the WebLogic Server installation and JVM. The tree in Figure 1 depicts the directory structure of a typical WebLogic domain.

Keep Your Skills Ahead of the Crowd

Keeping your IT skills ahead of the crowd is not as difficult as most people fear. Staying on top of the trends may seem like a daunting task if, like most people, you assume that each new technology is a completely new invention that you must learn from the ground up. Fortunately, nothing is really all that new. Inventors typically create new technologies by studying existing technologies, then building upon them in ways that extend and improve them. 100% new technological advancements are very rare.

Inventors almost always leverage legacy technologies as they invent new ones. Why not leverage your own knowledge of those legacy technologies as you try to learn about the new inventions? To learn about new technologies as painlessly as possible, consider how each new advancement is similar to what you already know.

For example, consider Web services. Web services are a new trend, but — at a technological level — the parts of a Web service are not all that unique. Web services are based on remote procedural calls — messages sent to a server, which calls the requested function. RPCs were developed years ago, and are hardly a new concept. Really, the only "new" thing in Web services is the standard that is being used to write the application. If you break down Web services in this way, it's easy to learn about them. To continue with this process, you might next explore the payload requirements, the process for determining what function to call, and how the call works. As you can imagine, it's a lot more efficient — and interesting — to learn about a new technology based on its relation to familiar technologies than to learn about it by reading the specification cover to cover.

As always, the devil is in the details. But most details are critical only if you want to specialize in a given technology. For instance, if you want to specialize in Web services, you need to familiarize yourself with the details of Web service development. In that case, your next step would be to learn how to format the messages, how to expose Web services, and so on.

*– **Adam Kolawa**, Ph.D.*
Chairman/CEO of Parasoft

SOAPtest®

It's automated. It's fast. And it's the most versatile Web Services testing tool.

SOAPtest® enables you to deliver better Web Services in less time.

The simple fact is, no other Web service testing product can do what SOAPtest can do. From verifying functionality and performance to ensuring security and interoperability, SOAPtest automates all critical testing processes across the entire lifecycle of your Web service.

But don't take our word for it.... Go to www.parasoft.com/soaptest_WDJ and try it for free—no commitment, no obligation. If you like it (and we suspect you will) just let us know. We'll be more than happy to help you and your development team get up and running.

For Downloads go to www.parasoft.com/soaptest_WDJ
Email: soaptest_WDJ@parasoft.com Call: 888-305-0041 x1209

PARASOFT®
We make software work.™

Features
- WSDL schema verification and compliance to standards
- Automatic test creation using WSDL and HTTP Traffic
- Data-driven testing through data sources (Excel, CSV, Database Queries, etc)
- Scenario-based testing through XML Data Bank and Test Suite Logic
- Flexible scripting with Java, JavaScript, Python
- WS-I Conformance: Basic Profile 1.0
- WS-Security, SAML, Username Token, X.509, XML Encryption, and XML Signature support
- WS-Security Interop testing emulator
- MIME Attachment support
- Asynchronous Testing: JMS, Parlay (X), SCP, and WS-Addressing support
- Windows Perfmon, SNMP, and JMX monitors
- Detailed Report generation in HTML, XML and Text formats
- Real-Time graphs and charts

Benefits
- Uniform test suites can be rolled over from unit testing to functional testing to load testing
- Prevent errors, pinpoint weaknesses, and stress test long before deployment
- Ensure the reliability, quality, security and interoperability of your Web service
- Verify data integrity and server/client functionality
- Identify server capabilities under stress and load
- Accelerate time to market

Platforms
- Windows 2000/XP
- Linux
- Solaris

Protocol Support
- HTTP 1.0
- HTTP 1.1 w/Keep-Alive Connection
- HTTPS
- TCP/IP
- JMS

Contact Info:
Parasoft Corporation
101 E. Huntington Dr., 2nd Flr.
Monrovia, CA 91016

www.parasoft.com

Converting the WebLogic-related property files

WebLogic Server 5.1 used a **"weblogic.properties"** file to configure applications.

In WebLogic Server 8.1, configuration is handled by a domain configuration file, **config.xml**, and by deploying descriptor files. Converting a "weblogic.properties" file to the config.xml file creates a WebLogic Server 8.1 domain for your applications and generates the XML files that define how your applications are set up.

The important things to consider for migrating a WebLogic property file from 5.1 to 8.1 are listed in table 4.

Location of the application-related property files

Every application – Web application or enterprise application – has its own property files that are meant to configure the application. These property files can be part of the application-related archive files (EARs or WARs) or they can be stored in the WebLogic Server 8.1 domain directory.

The WebLogic Server 8.1 domain directory will be the "Current Working Directory" for all applications deployed in a particular domain.

The property files that are common across different applications deployed in a particular WebLogic Server 8.1 domain can also be stored in the domain directory.

Modifying the Startup Scripts

When the WebLogic Server 8.1 domain is created using the "WebLogic Server 8.1 Configuration Wizard," four script files will be created:

– startWebLogic.cmd
– installService.cmd
– uninstallService.cmd
– setEnv.cmd

The "startWebLogic.cmd" script can be used to start the WebLogic Server in the console mode. This script is generally used for running the WebLogic server in development mode.

The "installService.cmd" script is used to install the WebLogic 8.1 administration server as a Windows NT service. The default Windows NT service name that's created when the "installService.cmd" script executes is "beasvc <domain name>_<server name>." Some tuning needs to be done in the "installService.cmd" script before installing the Windows NT service. Some of the changes that can be done are:

- Change the minimum and maximum memory to be used by the JVM for running the WebLogic 8.1 Server
- Add any JVM parameters that need to be passed
- Extend the classpath environment variable to include external classes or JAR files

Migration from JDK 1.3 to 1.4

WebLogic Server 5.1 was certified on JDK 1.3 whereas 8.1 is certified on JDK 1.4. As a result, all the application sources have to be migrated to JDK 1.4. The most commonly used method that's been deprecated in JDK 1.4 is the constructor for the "java.util.Date" class. The "java.util.Calendar" class has to be used to create the "java.util.Date" object.

Converting Applications to Enterprise Archives or Web Application Archives

Enterprise applications and Web applications could only be deployed in WebLogic Server 5.1 as exploded archive directories. They couldn't be deployed as archive files (EARs or WARs). With this approach, there was a good chance patches could be installed as individual class files or JSP files in the exploded archive directories. WebLogic Server 8.1 supports deployments that are packaged either as archive files using the Jar utility, or as exploded archive directories.

WebLogic Server 8.1 supports deployment of J2EE-specified standalone modules like Enterprise Java Beans and Resource Adapter modules. Standalone modules generally provide parts of a larger, distributed

TABLE 4

Migration Consideration	WebLogic Server 5.1	WebLogic Server 8.1
Privileged User Account	This account is configured in weblogic.properties and is used to start the WebLogic Server.	There's no Privileged User Account. All accounts are configured using WebLogic Administration Console.
SSL Configuration	The Private Key file, Server Certificate, and Trusted Certificates are maintained and configured as independent files. The Private Key file isn't protected.	The Private Key file and Server Certificate are secured in the Java KeyStore file. This way, the Private Key file is secure and can't be misused without knowing the KeyStore password and the Private Key password.
Enterprise Java Beans (EJB) Deployment	Enterprise Java Beans are deployed independent of the Enterprise application using the property "weblogic.ejb.deploy" in weblogic.properties.	Enterprise Java Beans can be deployed either as part of an Enterprise Archive Application (EAR) or as an independent component. One of the easiest ways to deploy either an Enterprise Archive or an EJB is through the Administration Console.
ConnectionPool and DataSource	One ConnectionPool can be associated with only one DataSource.	The configuration of the ConnectionPool and DataSource is separated in WebLogic Server 8.1. So, a specific ConnectionPool can be associated with one or more DataSources.
High Availability of ConnectionPools	ConnectionPool failover capability isn't supported in WebLogic Server 5.1.	ConnectionPool failover capability is provided in WebLogic Server 8.1 through MultiPools. MultiPools can be set up to include a set of ConnectionPools to provide either load-balancing or high-availability features for DB connectivity.
Password information in WebLogic Property File	Passwords stored in a WebLogic Server 5.1 property file were in plain text.	The password (for Database, Keystore, JMS, etc) stored in WebLogic 8.1 Server Domain property file are encrypted.
Deploying applications as either Web Application Archives (WARs) or Enterprise Application Archives (EARs).	Applications couldn't be deployed as EARs or WARs in WebLogic Server 5.1.	Applications can be deployed as EARs, WARs, or exploded directory structure in WebLogic Server 8.1.
Automatic Backup of WebLogic Property files	WebLogic Server 5.1 Property files aren't backed up automatically.	WebLogic Server 8.1 maintains backup copies of config.xml.

WebLogic Property File Migration considerations application, but don't necessarily provide a direct user interface.

WebLogic Server 8.1 supports deployment of all three types of J2EE applications, namely: Web applications, enterprise applications, and client applications.
Table 5 provides information on the constituents in each type of J2EE application that can be deployed in WebLogic Server 8.1.

Recommendations

- Deploy standalone Web applications, resource adapters, and EJBs as part of an enterprise application. It allows easier application migration, additions, and changes.
- Deploy applications in exploded archive format only if:
  - Partial updates are necessary for the application without redeploying the entire application
  - The administration console has to be used for editing the deployment descriptors
  - The application performs direct file system I/O through the application context
  - The application contains static files that are periodically updated (for example, the use of Brio Reporting scripts as part of the Web application)

- Package deployment files in an archive format (.ear, .jar, or .war) when distributing files to different users or environments.

Upgrading Enterprise Java Beans Applications

WebLogic Server 5.1 supported J2EE Enterprise Java Beans (EJB) version 1.1. WebLogic Server 8.1 supports both EJB v1.1 and EJB v2.0. This means that the 1.1 Beans used in WebLogic Server 5.1 can be deployed on WebLogic Server 8.1. But, it's best to migrate EJB 1.1 Beans to EJB 2.0 Beans before deploying them on WebLogic Server 8.1.

To migrate applications from WebLogic Server 5.1 to 8.1, it's best to migrate the EJBs from v1.1 to v2.0. These minimal changes have to be made for migrating EJBs from v1.1 to v2.0 on WebLogic Server 8.1:

- Update the reference to DTD in the J2EE deployment descriptor (WEB-INF/ejb-jar.xml) to "<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN' 'http://java.sun.com/**dtd/ejb-jar_2_0.dtd**'>"
- Update the reference to DTD in the WebLogic deployment descriptor (WEB-INF/weblogic-ejb-jar.xml) to "<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD WebLogic **8.1.0** EJB//EN" http://www.bea.com/servers/wls**810**/***dtd/weblogic-ejb-jar.dtd***" >"
- Change the INITIAL_CONTEXT_FACTORY property for JNDI Lookup to use "***weblogic.jndi.WLInitialContextFactory***"
- Change the PROVIDER_URL property for JNDI Lookup to use "t3://localhost:<port>"

Please refer to the J2EE specifications for EJB 2.0 and WebLogic documentation (http://e-docs.bea.com/wls/docs81/ejb/index.html) for the complete upgrade of EJBs from v1.1 to v2.0. The WebLogic Server DDConverter utility can also be used for converting EJB 1.1 Beans to EJB 2.0 Beans.

SSL Configuration Changes

The SSL Configuration has changed significantly from WebLogic Server 5.1 to WebLogic Server 8.1. Some of the important changes that have been made are:

- The Private Key file and the server certificate file can't be deployed as individual files in WebLogic Server 8.1. The Private Key file and the server certificate file have to be secured in a Java KeyStore before being deployed in WebLogic Server.
- The client certificate files can't be deployed as individual files in WebLogic Server 8.1. The client certificate files have to be secured in a Java KeyStore before being deployed in WebLogic Server.

The Private Key file in WebLogic Server 5.1 used to be in 'DER' format. It has to be converted to 'PEM' format before being imported into a Java KeyStore.
These are the steps involved in converting a Private Key file that's in 'DER' format into 'PEM' format and loading it into a new Java KeyStore file:

1. Copy the Private Key file xenon-key.der and certificate file xenon-cert.pem into <Drive>:\tmp\certs directory
2. Create a text file called <Drive>:\tmp\certs\header.txt file and store the following information. Note that the header.txt file should end with a new line.
   -----BEGIN RSA PRIVATE KEY-----
3. Create a text file called <Drive>:\tmp\certs\footer.txt and store the following information. Note that the footer.txt file should end with a new line.
4. -----END RSA PRIVATE KEY-----
5. Open a command window
6. Run <Drive>:\bea\wls810\user_projects\domains\WSDomain\setenv.cmd
7. Change the directory to <Drive>:\tmp\certs and execute
   java utils.der2pem xenon-key.der header.txt footer.txt.

   A new file named xenon-key.pem file has to be created in <Drive>:\tmp\certs directory.

– continued on page 45

TABLE 5

Deployment Unit	Scope	Components
Web Application	J2EE	WEB-INF/web.xml
		Servlets or JSP pages with helper classes
	WebLogic	WEB-INF/weblogic.xml
Enterprise Java Bean	J2EE	META-INF/ejb-jar.xml
		Java classes
	WebLogic	META-INF/weblogic-ejb-jar.xml
		META-INF/weblogic-cmp-rdbms-jar.xml
Connector	J2EE	META-INF/ra.xml
	WebLogic	META-INF/weblogic-ra.xml
Enterprise Application	J2EE	META-INF/application.xml
		Web Application(s)
		Enterprise Java Bean(s)
		Client Application(s)
		Helper Classes
	WebLogic	META-INF/weblogic-application.xml
Client Application	J2EE	META-INF/application-client.xml
	WebLogic	client-application.runtime.xml

J2EE application constituents that can be deployed in WebLogic Server 8.1

A Look Ahead to the Service-Oriented World

DEFINING SOA WHEN THERE'S NO SINGLE, OFFICIAL DEFINITION

By Thomas Erl

BEA recently announced that it is broadening its SOA consulting practice, and that it has created a tool companies can use to learn about SOA and figure out how prepared they are to transition to the new architectural model.

While BEA and other major vendors, such as IBM and Microsoft, continue to deepen their investments in SOA, many of us are still struggling to understand what SOA actually is.

How well do you know SOA? If you were asked to write a definition right now, what would it be? One of the challenges has always been to distinguish SOA from a standard distributed architecture that use Web Services. Another has been to pinpoint exactly what the well-publicized service-orientation paradigm includes.

To help with these questions, we asked noted SOA expert Thomas Erl to provide some clarity. Thomas wrote "Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services," *last year's top-selling book in both Web Services and SOA. He is releasing his second book, a 700 page effort entitled,* "Service-Oriented Architecture: Concepts, Technology, and Design," *later this year. Below is an excerpt from his upcoming book introducing us to the basics of SOA and service-orientation.*

Service-Oriented Architecture
Concepts, Technology, and Design
Thomas Erl

Service-Oriented Architecture

There is no single, official definition of service-oriented architecture. Because the term "service-oriented" has existed for some time, it has been used in different contexts and for different purposes. One constant through its existence, though, is that it represents a paradigm based on the concept of decomposition. Something, anything that can be better constructed or carried out or managed if it is done so as a well-defined collection of related units.

Service-orientation is therefore not always a technical paradigm. The Yellow Pages are full of service-oriented businesses. Individual companies are service-oriented in that each provides a distinct service that can be used by multiple consumers. Collectively, these businesses comprise a business community. It makes perfect sense for a business community not to be served by a single business outlet providing all services. By decomposing the community into specialized, individual outlets, we achieve an environment in which these outlets can be distributed.

Even in a distributed model, if we impose overbearing dependencies, we could inhibit the potential of individual businesses. Although we want to allow outlets to interact and leverage each other's services, we want to avoid a model in which outlets form tight connections that result in constrictive inter-dependencies. By empowering businesses to self-govern their individual services, we allow them to evolve and grow relatively independent from each other.

Though we encourage independence within our business outlets, we must still ensure that they agree to adhere to certain baseline conventions. For example, a common currency for the exchange of goods and services, a building code that requires signage to conform to certain parameters, or perhaps a requirement that all business employees speak the same language as the native consumers. These conventions standardize key aspects of each business for the benefit of the consumers without significantly imposing on the business's ability to exercise its self-governance.

When coupled with "architecture" service-orientation takes on a technical connotation. "Service-oriented architecture" is a term that represents a model in which automation logic is decomposed into

Author Bio:
Thomas Erl is president and chief architect of XMLTC Consulting, Inc. (www.xmltc.com), a consulting firm in Vancouver, Canada specializing in delivering service-oriented and XML-centric solutions. Thomas is the author of Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services as well as the upcoming Service-Oriented Architecture: Concepts, Technology, and Design (for more information visit www.serviceoriented.ws). He has published over 30 papers, and has established an integration framework for XML and Web Services. He has recently become a member of the OASIS SOA-RM Technical Committee.

Contact:
terl@xmltc.com

Nothing beats our racks Absolutely nothing

CARRIER CLASS DATA CENTERS

Highly Secure Guarded Facility
24/7/365 Network Operations Center
24/7/365 Technical Support
Redundant Conditioned Air Systems
Redundant Fiber Entry Points
Multiple Uninterruptible UPS Systems
Multiple 1250 KW Generators Onsite
VESDA Early Warning Smoke Detection

Robert Marsh, Head Surfer

START YOUR OWN WEB HOSTING BUSINESS TODAY!

from **$299***

Instant Activation!

Dedicated Server

Dual Xeon 2.4 GHz
2 GB RAM • 2 x 73 GB SCSI HD
Remote Console • Remote Reboot
2000 GB Monthly Transfer Included

Over 20,000 Servers!

1-800-504-SURF | ev1servers.net

PLESK7 RELOADED
Preferred Control Panel

IP Compliant. Price subject to change. Quantities Limited.
*Per month. Set-Up fees apply. See web site for complete details.

smaller, distinct units of logic. Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed.

Service-oriented architecture (SOA) encourages individual units of logic to conform to a set of design principles, one of which is to exist autonomously. This allows units to evolve independently from each other, while still maintaining a sufficient amount of commonality. It also results in a business automation environment with distinct characteristics and benefits.

SOAs can be extremely sophisticated. Current development tools and server platforms are constantly broadening the feature set and capabilities in support of the creation of service-oriented solutions. However, before we can discuss the many ways in which SOA can support enterprise-level automation, we must first look at SOA in its most fundamental form.

This primitive model establishes the core architecture that underlies all variations of SOA. It consists of three primary components that form a unique relationship to achieve automation.

Services

Every business process consists of a series of steps. Larger processes typically include one or more sub-processes that support the parent process. These sub-processes also consist of a series of steps that reside within a logical boundary. The boundary encloses a distinct task or function provided by the sub-process.

Services represent real-life actions. The size or scope of the action is not pertinent to the task or function being encapsulated within a service. What's relevant is that the boundary of the task or function be distinct. Therefore, a service can represent any part of a process. It is important to note that in doing so the service establishes a standardized entry point into the process's business logic.

An extension of this concept into a physical implementation environment establishes a service as a self-contained *unit of processing logic*. The service also has a distinct functional boundary, and is designed to perform a specific task. Its task may be elaborate or limited. For example, a service may execute a series of actions involving other services. Or, a service's sole function may be to provide access to a fixed resource, such as a repository. Regardless, its most fundamental characteristic is that it is relatively independent or *loosely coupled* from other services.

Loose Coupling

Within SOA, loose coupling represents the basis of a communications agreement between services. The agreement consists of an understanding that in order for services to communicate with each other, they must be aware of each other. This awareness is achieved through the use of *service descriptions*.

A service description, in its most basic format, establishes:

- the name of the service
- a description of the data expected by the service
- a description of any data returned by the service

For example, service A is aware of service B because service A has gained access to service B's service description. If service A could somehow communicate with service B without even knowing of service B's existence, there would be no communication agreement, and the services would be considered *decoupled*. (Decoupled communication is often made possible through the use of middleware or intermediary components that reside between two programs.)

An additional part of the loosely coupled agreement is that communication between services be self-contained as well. Meaning, each transmission of a *unit of communication* passed between services should occur independently from others.

So, having acquired service B's service description, service A communicates with service B. Service B receives the (unit of) communication but may not be obligated to respond to service A. Further, the communication itself occurred without any direct connection to service B. If service A had established a direct connection to service B, upon which it would have transferred data and received a response, the services would be considered *tightly coupled*.

Another characteristic of tight coupling is a dependency between units of processing logic. The logic is programmed in such a way that a change to one piece of logic could easily affect others that it references or that are referencing it. Therefore, if service A and B are tightly coupled, changes to one may require changes to the other. In a loosely coupled architecture, changes to either service would not affect the other as long as the original communication agreement (as expressed by the service descriptions) is preserved.

How do we achieve loose coupling? We need a communications framework that is based solely on the aforementioned "independent units of communication." This is where *messaging* comes in.

Messaging

Messages are a cornerstone technology of SOA. They are what implement and make many service-orientation principles possible, and form the basis for all inter-service communication. As with the concept of SOA itself, messaging-based communication is nothing new. It's been used by middleware products for years.

However, the preferred way to implement messaging in SOA is very specific. Once a service sends a message on its way, it loses control of what happens to the message thereafter. That is why we require *independent* units of communication to achieve true loose coupling. Messages, like services, need to be relatively self-contained. That means putting as much intelligence in a message as required. This includes the actual structure and typing of the message data.

Messaging provides SOA with the option of communicating synchronously or asynchronously. While SOA fully supports synchronous message exchanges, its emphasis on loose coupling and communication independence encourages asynchronous interaction scenarios. The net result is the ability to support a variety of communication models ("message exchange patterns").

Service-oriented messaging in contemporary environments relies on a sophisticated framework that supports the transmission and runtime processing of information-heavy messages. Messages can be equipped with a variety of composable features that can handle everything from security and reliable delivery to routing and the processing of polices.

Note how we just discussed the compo-

TANGOSOL™

A Cure for Downtime.

Eliminate the single points of failure, add fault tolerance and enable transparent failover for your J2EE applications.

TANGOSOL COHERENCE™ has no single points of failure, ensuring high availability and reliability for your mission-critical J2EE applications. Application state, HTTP sessions and data caches are never at risk. Applications using Coherence keep it up without interruption, even when servers go down.

Try before you buy.
Download a free evaluation license at

www.tangosol.com

Coherence™ is pure Java software that reliably and cost-effectively scales J2EE applications across any number of servers in a cluster. Coherence provides reliable coordinated access to in-memory data that enables safe data caching and user session management for applications in a clustered environment.

Sales	+1.617.623.5782 or sales@tangosol.com
Support	support@tangosol.com or http://www.tangosol.net
Pricing	US$1,995 to US$4,995 per Coherence production CPU Development licenses are offered free of charge.

nents of the primitive SOA model without referencing Web Services, WSDL, or SOAP. These technologies, of course, have become the most successful means by which to deliver service-oriented solutions. In today's SOA, services exist as Web Services, service descriptions are primarily realized through WSDL definitions, and messaging is standardized through the SOAP format. It is important to remember, though, that SOA, in a primitive form, is technology-agnostic. The same applies to the principles of service orientation.

Principles of Service Orientation

So far, we established that a fundamental SOA consists of a set of services that use service descriptions to remain loosely coupled and rely on messaging as a means of communication. These core characteristics are shaped and supported by the principles of service orientation, which form the basis for service-level design.

Earlier we established that there is no formal definition of SOA. There is also no single governing standards body that defines the principles behind service orientation. Instead, there are many opinions, originating from public IT organizations to vendors and consulting firms, about what constitutes service orientation. (See http://www.serviceorientation.org/ for more information.)

Service orientation is said to have its roots in a software engineering theory known as "separation of concerns." This theory dictates that it is beneficial to break down a problem into a series of individual concerns. By doing so, complexity is reduced and the overall quality of the collective concerns is increased. This theory has been implemented in different ways with different development platforms. Object-oriented programming and component-based programming approaches, for example, achieve a separation of concerns through the use of objects, classes, and components.

Serviceorientation can be viewed as a distinct manner in which to realize a separation of concerns. The principles of service orientation provide a means of supporting this theory while achieving a foundation paradigm for SOA as a distinct architectural model.

As previously mentioned, there is no official set of service-orientation principles. There are, however, a common set of principles most associated with service orientation. These are listed below and described in detail throughout this book.

- Services are reusable – Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.
- Services share a formal contract – In order for services to interact, they need not share anything but a formal contract that defines the terms of information exchange and any supplemental service description information.
- Services are loosely coupled – Services must be designed to interact on a loosely coupled basis, and they must maintain this state of loose coupling.
- Services abstract underlying logic – The only part of a service that is visible to the outside world is what is exposed via the service's description. Underlying logic, beyond what is expressed in this description, is invisible and irrelevant to service requestors.
- Services are composable – Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.
- Services are autonomous – The logic governed by a service resides within an explicit boundary. The service has complete autonomy within this boundary, and is not dependent on other services for it to execute this governance.
- Services are stateless – Services should not be required to manage state information, since that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- Services are discoverable – Services should allow their descriptions to be discovered by and understood by humans and service requestors that may be able to make use of their logic.

Of these eight, autonomy, loose coupling, abstraction, and the need for a formal contract can be considered the core principles that form the baseline foundation for SOA.

"Messages are a cornerstone technology of SOA. They are what implement and make possible many service-orientation principles, and form the basis for all inter-service communication"

Contemporary SOA

This introduction has only scratched the surface of this broad subject matter. The principles behind service orientation and the architectural paradigm established by SOA can be applied throughout an enterprise. When married with the open Web Services technology framework, it establishes an enormous benefit potential for increasing organizational agility and improving the lines of communication across previously disparate environments. Once taken to this level, a service-oriented architecture evolves into what we classify as "contemporary SOA." Implementing this comprehensive and complex model can significantly impact an organization. Business modeling, resource allocation, and technical infrastructure are just some of the areas that can be affected. The remainder of this book is dedicated to exploring the many aspects of contemporary SOA.

This has been an excerpt from: *Service-Oriented Architecture: Concepts and Technology* by Thomas Erl, approximately 700 pages, ISBN: 0131858580, Prentice Hall/PearsonPTR, Copyright 2005. (For more information, see www.serviceoriented.ws/.)

What//: are your Web applications saying to your customers?

diagnoSys

Are your Web applications open 24 hours? Get the tips and best practices you need to keep your apps running at peak efficiency 24 hours a day with WebApp News, the free newsletter from the performance experts at H&W.

Sign up today at:
www.hwcs.com/wldj08.asp

You might as well be closed for business if your Web applications aren't running at top speed 24 hours a day. With poor Web performance costing businesses $25 billion a year, you need to find problems, and you need to find them fast.

You need DiagnoSys.

DiagnoSys is the first intelligent performance management solution that analyzes your J2EE and .NET applications from end to end. DiagnoSys starts analyzing quickly, gathers data directly from all application resources, and proactively finds and helps fix the real cause of problems – before they cost your business big time.

So when it's absolutely essential that your Web applications say "Open 24 hours," trust DiagnoSys.

SILVER ANNIVERSARY 25 1979 - 2004

H&W
www.hwcs.com | 1.800.338.6692

Bridging the Enterprise Computing Gap For 25 Years

Scheduling Jobs in WebLogic Server

WRITING APPLICATION-SPECIFIC SCHEDULED JOBS

By Vijay Chinta & Thomas Kunnumpurath

Author Bios:
Vijay Chinta is an enterprise architect at Nokia. His primary area of expertise is J2EE development for enterprise applications in the fields of telecommunication and transportation.

Thomas Kunnumpurath is a senior software engineer at Nokia. He has several years experience in Java/J2EE application development.

Contact:
vijay.chinta@nokia.com
kj.thomas@nokia.com

The need to run scheduled jobs is increasing and becoming common across all J2EE applications. The current J2EE specification doesn't provide an easy way to schedule jobs inside an enterprise application.

We can broadly classify scheduled J2EE jobs into two categories: Server-specific scheduled jobs and application-specific scheduled jobs. This article explores how to schedule application-specific jobs inside a WebLogic Application Server.

Server-Specific vs. Application-Specific Scheduled Jobs

Server-specific scheduled jobs are associated with the lifecycle of the application server. Typically these jobs are started during the server's startup and will be active until it's shut down. As a consequence, all the resources needed for scheduled jobs have to be specified in the server's classpath.

Application-specific scheduled jobs are associated with the lifecycle of the enterprise application. They are started after the application is deployed and will be active until the application is undeployed. The resources required for application-specific scheduled jobs are packaged in the enterprise application (EAR – enterprise archive resource).

Application jobs are preferable to server jobs for the following reasons:

- Any changes to the application jobs can be redeployed easily with the application (EAR file). For server jobs the server has to be restarted for the new changes to take effect, and this may not be desirable as it impacts the availability of other applications deployed on that server.
- Since the scheduled jobs usually invoke business functionality, it's more logical to package them with in the EAR file.
- By associating the resources with the EAR file, we can deploy the EAR file on another instances of WebLogic Server without having to reconfigure the resources for that server.

Schedule Jobs Inside an Enterprise Application

We will look at an example of how to implement application-specific scheduled jobs. Here we schedule two jobs that invoke business functions at certain specified intervals.

- Order submission - invoked every day.
- Inventory submission - invoked once a week.

In this example we're going to use WebLogic's

Application Lifecycle Events in tandem with WebLogic's Timer Notifications.

These are the three steps involved in setting up scheduled jobs:

1. Implement the Timer Notification Listener, which is responsible for adding, listening, and handling the timer notifications.
2. Implement the Application Lifecycle Listener, which instantiates the Timer Notification Listener on desired application lifecycle events.
3. Register the Application Lifecycle Listener in weblogic-application.xml

Implement the MyAppJobScheduler

MyAppJobScheduler will implement the javax.management. NotificationListener interface.

```
public final class MyAppJobScheduler implements NotificationListener
```

This class will instantiate weblogic.management.timer.Timer class and register itself as a listener for timer notifications. The Timer class is a WebLogic implementation of javax.management. TimerMBean.

```
timer = new Timer();
timer.addNotificationListener(this, null, "some handback object");
```

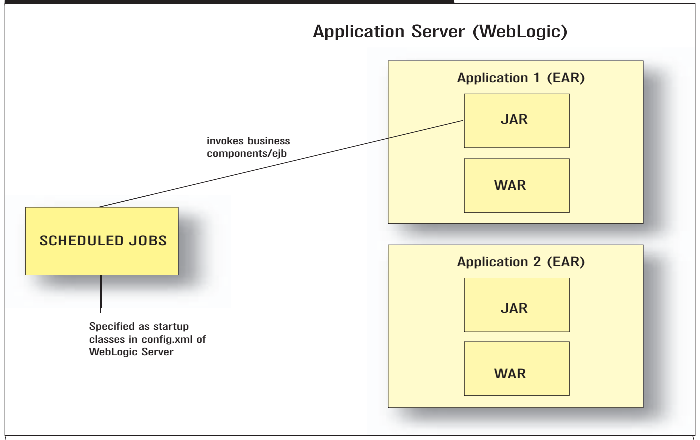
The order notification and inventory notification are added to the timer before the timer starts.

```
timer.addNotification("OrderSubmission", "Order Submission", this,order
SubmissionDate, DAILY_PERIOD);
timer.addNotification("InventorySubmission", "Inventory Submission",
this,inventorySubmissionDate, WEEKLY_PERIOD);
```

We will schedule the jobs so the Order job will run every day at 5 p.m. and the Inventory job will run once a week, every Friday at 10:30 p.m. Please check the code listing for details on how to construct the Date object with the specified time.

The callback method will get the notification. Based on the notification, a business process component (EJB) can be invoked to fulfill the scheduled job.

FIGURE 1


Application Server (WebLogic)
Application 1 (EAR)
JAR
WAR
Application 2 (EAR)
JAR
WAR
invokes business components/ejb
SCHEDULED JOBS
Specified as startup classes in config.xml of WebLogic Server

Server Jobs invoking the business functionality of Application 1

```
public void handleNotification(Notification notif, Object handback)
{
    String type = notif.getType();
    if ( type.equals("OrderSubmission") )
    {
       // invoke the component or ejb that does the order processing and
submission
    }
    else if ( type.equals("InventorySubmission") )
    {
        // invoke the component or ejb that does the inventory processing
and submission
    }
 }
```

The cleanUp method of MyAppJobScheduler stops the timer, and removes all the notifications that were added to it. We can invoke the cleanUp method as shown above from the preStop method, or it can be invoked from the finalize method of MyAppJobScheduler.

```
public synchronized void cleanUp()
{
    System.out.println(">>> MyAppJobScheduler cleanUp method called.");
    try
    {
       timer.stop();
       timer.removeNotification(orderNotificationId);
       timer.removeNotification(inventoryNotificationId);
       System.out.println(">>> MyAppJobScheduler Scheduler stopped.");
    }
    catch (InstanceNotFoundException e)
    {
       e.printStackTrace();
    }
}
```

Here is how the final method of MyAppJobScheduler will look:

```
  protected void finalize() throws Throwable
  {
            System.out.println(">>> MyAppJobScheduler finalize called.");
            cleanUp();

            super.finalize();
  }
```

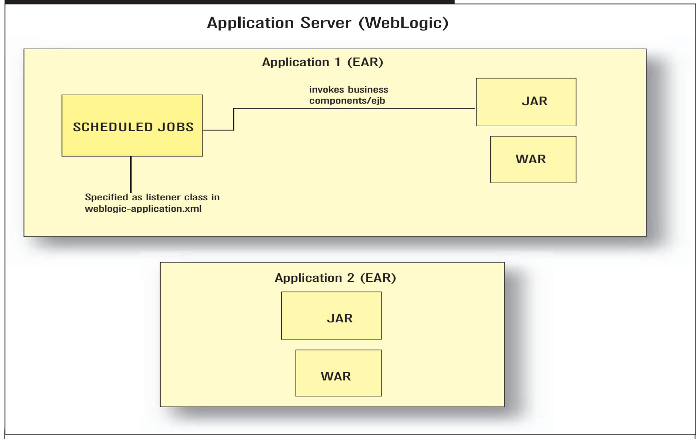
Implement MyAppListener

We will write a class MyAppListener that extends weblogic.application.ApplicationLifeCyleListener. Application lifecycle listener events provide handles that developers can use to control behavior during deployment, undeployment, and redeployment.

```
  public class MyAppListener extends ApplicationLifecycleListener
```

We want to start the scheduled jobs as soon as the application is deployed. So we will invoke the MyAppJobScheduler in the postStart method of MyAppListener. If you want to invoke the Scheduler before the startup, the invocation will be in the preStart method.

FIGURE 2


Application Server (WebLogic)
Application 1 (EAR)
invokes business components/ejb
SCHEDULED JOBS
JAR
WAR
Specified as listener class in weblogic-application.xml
Application 2 (EAR)
JAR
WAR

Application Jobs invoking the business functionality of Application 1

```
public void postStart(ApplicationLifecycleEvent evt)
{
          System.out.println( “MyAppListener:postStart Event”);

          //Start the Scheduler
          myAppJobScheduler = new MyAppJobScheduler();
}
```

We want to unschedule the jobs before the application is undeployed. So we invoke the MyAppJobScheduler’s cleanUp method in the preStop method of MyAppListener.

```
public void preStop(ApplicationLifecycleEvent evt)
{
  System.out.println( “MyAppListener:preStop Event”);

  //Stop the Scheduler
  myAppJobScheduler.cleanUp();
}
```

Register MyAppListener

In the weblogic-application.xml, register MyAppListener class for application lifecycle events.

```
<listener>
   <listener-class>MyAppListener</listener-class>
</listener>
```

Include MyAppJobScheduler, MyAppListener in the application classpath, or alternatively you can specify the jar file using <listener-uri> parameter.

```
<listener>
   <listener-class>MyAppListener</listener-class>
   <listener-uri>scheduler.jar</listener-uri>
</listener>
```

References

- *WebLogic Server:* Programming Application Lifecycle Events: http://e-docs.bea.com/wls/docs90/programming/lifecycle.html
- *WebLogic Timer:* http://e-docs.bea.com/wls/docs90/javadocs/weblogic/management/timer/Timer.html
- *J2EE Notification Listener:* http://java.sun.com/j2ee/1.4/docs/api/javax/management/NotificationListener.html

```
MyAppJobScheduler.java
/*
 * MyAppJobScheduler.java
 *
 */

import java.util.*;
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.InstanceNotFoundException;
import weblogic.management.timer.Timer;

// Implementing NotificationListener
public final class MyAppJobScheduler implements NotificationListener
{
    private static final long DAILY_PERIOD = Timer.ONE_DAY;
    private static final long WEEKLY_PERIOD = 7 * Timer.ONE_DAY;

    private Timer timer;
    private Integer orderNotificationId;
    private Integer inventoryNotificationId;

    public MyAppJobScheduler()
    {
        // Instantiating the Timer MBean
        timer = new Timer();
```

```
        // Registering this class as a listener
        timer.addNotificationListener(this, null, “some handback
         object”);

        // These values should be read from a property file
        int orderSubmissionHour=17;
        int orderSubmissionMinute=0;

        int inventorySubmissionDay = 6;
        int inventorySubmissionHour = 22;
        int inventorySubmissionMinute = 30;

        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.HOUR_OF_DAY,orderSubmissionHour);
        calendar.set(Calendar.MINUTE,orderSubmissionMinute);
        Date orderSubmissionDate = calendar.getTime();

        calendar.set(Calendar.DAY_OF_WEEK, inventorySubmissionDay);
        calendar.set(Calendar.HOUR_OF_DAY,inventorySubmissionHour);
        calendar.set(Calendar.MINUTE,inventorySubmissionHour);
        Date inventorySubmissionDate = calendar.getTime();

        // Add the order notification which should run every day at
            5:00 PM
        orderNotificationId = timer.addNotification(“OrderSubmission”,
                           “Order Submission”, this,orderSubmissionDat
```

```
e, DAILY_PERIOD);

        // Add the inventory notification which should run every Friday
           at 10:30 PM
        inventoryNotificationId = timer.addNotification
         ("InventorySubmission",
                         "Inventory Submission", this,
                          inventorySubmissionDate, WEEKLY_PERIOD);

        timer.start();
        System.out.println( ">>> MyAppJobScheduler started." );

    }

    protected void finalize() throws Throwable
    {
       System.out.println(">>> MyAppJobScheduler finalize called.");
       cleanUp();

       super.finalize();
    }

    public synchronized void cleanUp()
    {
       System.out.println(">>> MyAppJobScheduler cleanUp method
        called.");
       try
       {
          timer.stop();
          timer.removeNotification(orderNotificationId);
          timer.removeNotification(inventoryNotificationId);
          System.out.println(">>> MyAppJobScheduler Scheduler
           stopped.");
       }
       catch (InstanceNotFoundException e)
       {
          e.printStackTrace();
       }
    }

    /* callback method */
    public void handleNotification(Notification notif, Object handback)
    {
       String type = notif.getType();
       if ( type.equals("OrderSubmission") )
       {
          // invoke the component or ejb that does the order processing
             and submission
       }
       else if ( type.equals("InventorySubmission") )
       {
          // invoke the component or ejb that does the inventory
              processing and submission
       }
    }

    public static void main(String[] args)
    {
       MyAppJobScheduler myAppJobScheduler = new MyAppJobScheduler();
    }
```

```
}

MyAppListener.java
/*
 * MyAppListener.java
 *
 */

import weblogic.application.ApplicationLifecycleListener;

import weblogic.application.ApplicationLifecycleEvent;

public class MyAppListener extends ApplicationLifecycleListener
{
  MyAppJobScheduler myAppJobScheduler;

  public void preStart(ApplicationLifecycleEvent evt)
  {
     System.out.println("MyAppListener:preStart Event");
  }

  public void postStart(ApplicationLifecycleEvent evt)
  {
     System.out.println( "MyAppListener:postStart Event");

     // Start the Scheduler
     myAppJobScheduler = new MyAppJobScheduler();
   }

  public void preStop(ApplicationLifecycleEvent evt)
  {
    System.out.println( "MyAppListener:preStop Event");

     // Stop the Scheduler
     myAppJobScheduler.cleanUp();

  }

  public void postStop(ApplicationLifecycleEvent evt)
  {
     System.out.println( "MyAppListener:postStop Event");
  }

  public static void main(String[] args)
  {
     System.out.println( "MyAppListener:main method");
  }

}

weblogic-application.xml

<weblogic-application>
   <listener>
      <listener-class>MyAppListener</listener-class>
    </listener>
</weblogic-application>
```

What's Wrong with Web Applications

AND WHAT TO DO ABOUT IT

By Channing Benson

Criticizing something as wildly successful as the World Wide Web seems a bit radical and potentially unpopular. There is no doubt that Tim Berners-Lee's elegantly simple invention enabled an unprecedented revolution in the way computers are used and by whom.

The amount of information currently accessible to anyone with an Internet connection is truly mind-boggling, and I do not think it is an exaggeration to say that the Web is the most revolutionary invention since the automobile in terms of the effect that it has had on how we live our lives.

So what brings me to gore this sacred cow? How can I be negative about something that everyone acknowledges as a technological marvel?

To be honest, most Web applications are distinctly user-unfriendly. The Web's undisputed power to access and present information has been mistaken for a universal entry point to computing and information resources. The view that every interaction between user and computer is a nail to be banged in by the hammer of the Web browser and server has been a step backward for both users and developers. Users have to live with horribly restrictive modal form-based solutions stripped of powerful UI paradigms common in conventionally delivered software. For instance, drag and drop and the ability to save application state in a locally stored document are not available. Developers suffer because Web-based architectures divide an application along illogical boundaries, which complicate the creation of a seamless user experience, and require the use of multiple divergent programming languages and frameworks.

These problems are primarily the result of taking a great idea and stretching it beyond the scope for which it was intended. The Web was initially about access to static content. It really wasn't much different from existing hypertext systems, except that the content was transparently distributed on the Internet – a small difference in implementation, a huge difference in results. Suddenly, authors could easily incorporate into their own works other material from a vast virtual library.

However the rapid adoption of the Web and the lure of its commercial potential made this technology newcomer the apparent solution to all the problems in the world. It didn't happen immediately, but it wasn't long before the Web browser became more than just a viewer, but a generic application deployment platform. Slowly but surely, dedicated client programs in traditional three-tier architectures were replaced by interfaces implemented as Web pages. There are some valid reasons for making this transition, but in the rush to migrate seemingly every application to a browser-based model, decision-makers threw the baby out with the bathwater. The fact of the matter is that HTML makes a very poor user interface layer. Even when tarted up – as it must be to even be functional – with Javascript or other client-side scripting solutions, browser-based interfaces fall far short of creating a rewarding experience for the user.

In addition to shortchanging the end user, webifying an application puts a heavy load on the developer, who suddenly must contend with a multitude of thorny issues in the areas of portability, state-management, multiple implementation languages, client-side vs. server-side operations, and security. This article examines these problems for users and developers in detail, with examples of the pros and (mostly) cons of Web-based applications. I'll conclude with suggestions for alternatives that include the positive aspects of Web applications, but without the downsides.

I am not arguing that all Web-based applications are clumsy and evil and should be eliminated. Particularly when you wish to make your application available to the entire Internet world and there are no existing standards or protocols for that applica-

Author Bio:
Channing Benson is a senior technical consultant at Hewlett-Packard. He works with HP's software partners in the areas of Java performance, J2EE architectures, and Itanium migration. A 20-year industry veteran, his previous areas of expertise include Lisp and X11/Motif. His recreational interests are music, snowboarding, back-country skiing, and Ultimate Frisbee.

Contact:
chanthing@hotmail.com

Want to spend more time developing software and less time supporting it? Spend some time discovering hp OpenView—a suite of software management tools that enable you to build manageability right into the applications and Web services you're designing.

Find out how the leading-edge functionality of hp OpenView can increase your productivity.

http://devresource.hp.com/d2d.htm

invent

tion, choosing to deliver on the Web may be the only choice. I am primarily concerned with the webification of applications provided by corporate IT departments, such as those for submitting travel expense reports, managing benefit packages, or reporting defects.

First though, let's examine an application that falls outside that category.

Before the World Wide Web, there was Usenet, an Internet-based system for discussion groups. Information was exchanged between server systems via the Network News Transport Protocol (NNTP). The beauty of Usenet was that it did not specify a particular client interface. Users could choose among a variety of "news readers" according to their particular preferences. Depending on the client program, posts could be made with the editor of the user's choice. Emacs users could even read news and post responses from within the editor. It was the best of both worlds: a standard that allowed the open exchange of information with people all over the world, but that also allowed users the freedom to interact with that system in the manner they found most productive.

Usenet has not gone away; there are now even Web-based news readers such as the one provided by Google. But now, forums that once would have been well served by Usenet have been moved to discussion systems that are accessible only via a particular Web site. If you want to participate, you have to use the interface provided by that Web site. You will have to create a new user account if you want to contribute to discussions and you will have to edit your posts with a somewhat-less-than-full-featured standard browser text box. So much for progress.

There are political reasons service providers might prefer the closed Web-based solution. It allows them to identify their users and to control and monitor access to the site. But these advantages to the provider of the service come at the expense of a poorer experience for the users of the service. Similar trade-offs can be seen in the evolution of travel expense report applications. At one company, paper travel expense forms were replaced by a standardized spreadsheet that the user would complete, print, sign, and submit.

It was a simple solution that gave the user power and flexibility. She had the full interface of a standard spreadsheet program at her disposal, the ability to save partially completed forms as well as save local copies of previously submitted forms. Spreadsheet fanatics, if so inclined, could graph their daily expenses, while the tabular presentation of data made it easy for mortals to notice if they forgot to include the charge for Wednesday's lunch. This document-centric scheme has since been replaced by a Web-based application that automates the logistical aspects of expense reporting. Report submission and approval are now done electronically via a Web-based application that routes reports through the designated approval chain. From a bureaucratic perspective, things are much neater. Managers no longer have to shuffle paper, bean counters get a centralized view of the process at all stages, and even the lowly employee submitting the report gets an easy way to track it through to reimbursement.

Unfortunately, the initial creation of the report is cumbersome, forms-based tedium, where each expense has to be entered one at a time. And after the expenses are entered one by one in isolation, instead of a nice calendar-based display, the user sees only a linear list of expenses to be included in the report. The interface itself provides practically no support to the user for organizing expenses, which again leads us to this irony of supposed progress. Yes, the Web-based application automates and centralizes the process. But again, it does so at the expense of crippling the end-user with an unhelpful, inflexible, and arcane user interface.

Why does this seem to be the inevitable result of webifying an application? Because the Web is a horribly inadequate user-interface solution platform, crafting a full-featured Web application that functions correctly is an exercise in cunning and frustration for the developer, while the end result typically tests a user's patience. Let's look at the restrictions that make it so.

There are two primary obstacles. The first one is that the Web is based on a stateless protocol. The predominant model of application design for the last two decades has centered on a document (i.e., a collection of state information) that the application operates on. In a conventionally programmed application, the full strength of the particular programming language employed can be brought to bear on the problem of representing the data being operated on. In Web-based applications, the programmer has a choice of several less-than-optimal techniques for maintaining state: fields (some hidden) in forms stored on the Web page itself, or server-side state maintained by various tricks for each user session or request. Each has particular drawbacks.

Values stored in forms-based storage might not be propagated between pages and can be viewed and possibly altered by clever users, hence posing security risks. Server-side storage may not scale well, or the techniques used to correlate a particular user with a particular set of program state, like cookies, may be disabled by the user's browser. Further, accessing the state information is not entirely transparent to the programmer and available storage resources on the client computer will probably go unused because the computations requiring the state occur on the server system. Finally, depending on the cleverness of the Web site developer, the document metaphor familiar to the user goes entirely out the window. For instance, there is typically no way to save multiple shopping carts for the same online vendor, the way a conventional word processor could save two versions of my annual letter to Santa.

The second hurdle is the fact that HTML and HTTP are technologies oriented entirely around the concept of the page. Rather than directly manipulate the user-interface components displayed to the user, the Web server has to render a page based on the last user request. Hence, even the simplest operation – like scrolling to the next page of a list of items – entails a round-trip to the server. The URL page request paradigm just does not map well to an interactive application. This is perhaps an oversimplification, but imagine telling an application developer of the late 80s or early 90s that any possible state or screen of the application had to be

Post-launch is NOT the time to be verifying web applications.

The wild blue yonder of operational monitoring and management is extremely unforgiving. Which means that going live with the monitoring software you used in development is a great way to go dead—quickly! You simply can't support operations if your staff is drowning in details provided by development profiling tools and debuggers. **Let NetIQ cover your apps...with AppManager.**

AppManager—the industry's easiest-to-use Systems Management suite—is a proven management system for monitoring J2EE application servers, databases, operating systems and even end-user response time. NetIQ's AppManager monitors ALL application components—not just your server. **NetIQ. Nobody does UNIX better. Nobody.**

Visit us at www.netiq.com/solutions/web to learn how we can help you address the challenges of your operational monitoring and management.

netiQ
Work Smarter.

addressable by a string (i.e., URL).

Now it is true that a vast amount of work has been done to provide frameworks and utilities that abstract that task away from the direct responsibility of the programmer. For instance, WebLogic Workshop's Struts-based Java Page Flow architecture simplifies and reduces the work required to implement the series of pages a user traverses to achieve a particular task. It's way easier than implementing the form and state management logic anew for every single application. However toolkits and frameworks, while simplifying life for programmers, don't make the inherent problems go away. They still show up where browser bookmarks are added in the middle of a multiple page state-dependent sequence, in accidental multiple presses of a submit button, in refreshes of a page that resulted from a POST rather than a GET. I am certain any experienced Web developer can add to the list. Think of how much more productivity could be on tap if framework developers didn't need to compensate for such a crippled application model.

A further consequence of using the Web as an application platform is the ever-present browser/server dichotomy. Can a particular functionality (such as form validation) be provided by the browser or must the logic reside on the server? Again, lots of programming effort has been expended to attempt to add user interface functionality to the browser. Most contemporary Web applications would be nonstarters without the availability of client-side scripting tools like JavaScript that were not even part of the original conception of the World Wide Web. Just as with server-side page generation frameworks, there is a wide range of solutions available for this problem: ActiveX components, VBScript, Java applets, Flash, and DHTML.

While undoubtedly powerful, these tools complicate matters for the developer. To begin with, they entail possible compatibility issues. If you decide to use VBScript, you've suddenly excluded any user not running Internet Explorer. Java Applets are great, giving you a "real" language in which to program your user interface, but what happens when the user's browser doesn't run a compatible JVM or she doesn't have the proper Java plug-in installed?

Beyond compatibility issues, these client-side technologies complicate development project roles. While graphic designers might have responsibility for the purely HTML aspects of a Web page, we can't count on them to have the programming chops to embed the JavaScript required to populate a SELECT item with the desired options. Truth be told, we can't expect even trained programmers to always understand the incantations necessary to achieve the desired effect via client-side scripting, particularly when the project requires the interface to be functional whether or not JavaScript is enabled on the user's browser. Judging from the number of forums and "tip-sheets" dedicated to the intricate tricks of getting even the simplest UI functionality to operate in the Web environment, it is a rare programmer indeed who is fully competent in all of the tricks of the trade. JSPs and tag libraries were supposed to help separate the roles of graphic designer and programmer, but the necessity of resorting to some form of client-side wizardry to create even the most rudimentary interface defies that solution.

Even if one uses the browser plug-in technologies that provide for a full-featured interface such as Flash and Java applets, besides the above-mentioned potential browser compatibility problems, these applications will operate in isolation from other similar Web applications. For instance, there's no dropping an object from a Java applet onto a Flash application and having something meaningful happen. Such functionality might be possible with Active-X plug-ins, but they are proprietary solutions that only work on Windows platforms and configuring them to allow such interoperability could quite easily create security issues for the client computer.

All right. Enough complaining already. Even if you don't necessarily agree with all of my arguments, I hope you at least grant that the average user of Web applications won't ever rave about the Web user interface the way someone might that of a conventionally programmed application like iTunes or Photoshop. Does it have to be that way? Is there not a path that can give us the connectedness of the Web along with a powerful, intuitive user interface?

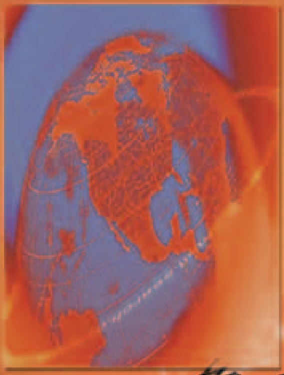
We want technology to enable the creation of client programs with full-featured user interfaces that can access server resources over the Internet. This is nothing new; client server architectures were the hot technology prior to the advent of the Web, although client server applications were primarily deployed on intranets rather than the Internet. So how might we incorporate Web concepts to advance the technology?
Some of the benefits of Web applications that would be desirable to retain are:

1. *Zero administration*. Web users don't have to install anything to run their applications. Updates are transparent.
2. *Flexibly networked*. For instance, an application for planning travel should be able to access flights, hotels, and car rentals from multiple vendors.
3. *The ability to render and display* HTML. While I have complained vociferously about the restrictions of HTML and HTTP, they are both firmly entrenched and there are application areas, such as online help, that would benefit from a modularized way of displaying HTML content.

There are no technical reasons that prevent the development of solutions that address these requirements. In fact, technologies exist today that address these needs; they're just underutilized in this context. I will discuss two possible approaches: first, thick clients written in Java and delivered over the network, and secondly, AJAX (Asynchronous JavaScript And XML), a relatively new technology that transfers processing power to the client-side (though still within a browser) and also mitigates the need to refresh an entire Web page for each transfer of data from the server.
For the dedicated client path, requirement number 1 is fulfilled by Sun's Java Web Start and its underlying Java Network Launch Protocol (JNLP). For some reason, this sim-

"Users deserve modern user interfaces for the programs they rely on, not forms-based tedium more appropriate for the days of block terminals"

BEA's Workshop can be even more amazing for the phone.



One Application.
Web. Voice. It's Easy.

AppDev™ VXML
for BEA's WebLogic™ Workshop

Delivering Web applications to phone users is as easy as clicking a mouse.

For additional information:

SandCherry, Inc.
1715 38th Street
Boulder, CO 80301

+1 (720) 562-4500 Phone
+1 (866) 383-4500 Toll Free
+1 (720) 562-4501 Fax

Info@sandcherry.com
www.sandcherry.com

Download
30 Day Free Trial
www.sandcherry.com

SandCherry™

ple and effective solution has never really caught on. It's been available for over four years and is now a standard part of the Java runtime.

You can read all about it at the Java Web Start home page (http://java.sun.com/products/javawebstart/index.jsp), but in a nutshell, you package your application into a JAR file or files, write a JNLP descriptor file and an HTML file with a link to that descriptor file, and then when the user of a properly configured browser (MIME types you know) clicks on that link, Java Web Start fires up and invokes the application. The JAR files are downloaded and executed on the local client machine. Henceforth, running the application does not require re-downloading the JAR files unless the application has been updated and the user requests the new version.

By default, Java Web Start applications run in a secure environment similar to the Java applet sandbox, but there are JNLP APIs for use by applications that need to access the local clipboard or disk. Applications that are implemented by signed JAR files can run in an unrestricted environment (once the user approves).

It is a mystery to me why Java Web Start has not gained more traction with the developer community. Java is a powerful, easy language for creating full-featured applications that are highly portable and Java Web Start is an easy to use tool for deploying such applications.
So, no excuses for not having Requirement 1 fulfilled.

What about Requirement 2? You'd have to be dead or deaf to not be aware of all the work that's been done over the past few years in the area popularly called "Web services" or "service-oriented architectures." The XML-based technologies of UDDI, WSDL, and SOAP have made possible a sort of uber-RPC for network clients where RPC servers with various functionalities can be discovered and those functionalities accessed. While in my opinion the reality in this arena has not matched the hype, there is no doubt that the potential is there. Google has WSDL and SOAP-based APIs available as a beta (so it is somewhat disappointing that their desktop search tool is based on HTML pages rather than a thick client).

The big obstacle here is the standardization of APIs for given functionalities. UDDI and WSDL describe functionality and how to access it, but if two different travel vendors (for instance) use different models for their Web services, it is a lot of work for the client developer to account for the differences in each and every provider they want to access. These issues are being addressed, but how quickly industries will define and adopt standard interfaces is unknown.

So, it would seem that Requirement 2 is partially fulfilled with more promise for the future. Incidentally, for an interesting report on combining Java Web Start and Web service technologies, see Allan Poda's devx.com paper entitled "Leverage JNLP and SOAP for Java Thick-client Development" (www.devx.com/Java/Article/22537), where he describes his experiments migrating a previous Web-based interface to a Java thick-client interface.

Requirement 3 (for some sort of HTML rendering capability within the thick client) is, as they say, "a simple matter of software." There are commercial products available, such as ICEBrowser and ICEReader from ICESOFT. There is also a number of freely available implementation including the barebones Swing HTMLEditorKit built into the Java runtime. Capabilities of these components obviously vary. You can find a good rundown of several freely licensable implementations in the article "Java Sketchbook: The HTML Renderer Shootout, Part 1" at http://today.java.net/pub/a/today/2004/05/24/html-pt1.html.

We haven't touched on requirements for security. Obviously this is vital, particularly for apps that are to be delivered over the greater Internet. Component code-signing techniques provide some measure of safety for downloading and running code, but users need to be educated to make good decisions when faced with the dialog that asks "Do you want to install and run: Bizmumble Signed and distributed by: So and So corporation?"

A full discussion of the security requirements and infrastructure for Web services would take an entire book (or two) of its own. The architects of Web services technologies understand the importance of security, and so it has not been neglected in the design and implementation of the technologies that make up Web services. Understanding the implications and necessities for a truly secure Internet application might be daunting, but not impossible.
If you want to stick with a pure browser-based solution, AJAX is a clever conglomeration of existing technologies that while not perfect, smoothes over many of the UI bumps. While the technology has been around for several years, it is getting new attention due to its use in several Google Web applications including Gmail, Google Maps, and Google Suggestions. A complete description is beyond the scope of this article, but essentially it is sophisticated JavaScript programming that takes advantage of XMLHttpRequest objects that make asynchronous server requests and update user interface components based on the results of these asynchronous queries. It eliminates the need to refresh the entire page simply to update one form element. For more info, try Googling "ajax Web interface" or see Jesse James Garret's excellent overview at http://adaptivepath.com/publications/essays/archives/000385.php.

So, from the examples presented, it is clear that the technologies required to replace a clunky HTML-based Web application with an application having a more full-featured user interface exist today. There may be other options in addition to the solution technology examples I provided. I believe that the goal of Microsoft's .NET is to realize a sort of "programmable Web" that would include thick client applications.

I grant that not all of these pieces that enable networked application interfaces superior to those of browser-based applications are fully formed and developed. But the groundwork is being laid. I hope that unlike the relatively weak adoption of Java Web Start, that the story for Web services and thick clients is a different one. Users deserve modern user interfaces for the programs they rely on, not forms-based tedium more appropriate for the days of block terminals. Developers and service providers need to start thinking outside the browser box and consider the benefits of thick network clients that use Web services, or at least think about enriching the user experience via innovative technologies like Ajax. They could get a much-needed boost in that direction if suppliers of the supporting infrastructure, like BEA, would move to provide support for these technologies in their development tools and product paradigms. In this way, everyone wins: users get more usable applications, developers get the credit, and tool vendors create pull for the new versions of their products by including innovative new features.

OSBC | OPEN SOURCE BUSINESS CONFERENCE

APRIL 5-6, 2005

THE WESTIN ST. FRANCIS, SAN FRANCISCO

osbc.com | THE BUSINESS OF OPEN SOURCE

osbc.com
THE BUSINESS OF OPEN SOURCE

OSBC is the only conference that focuses on the business of open source strategies and solutions and what they mean to your company. This April, join top level executives and industry luminaries who are paving the way in the Open Source arena.

LEARN to build sustainable, higher-margin Open Source businesses.

HEAR directly from business leaders about how to start and sustain a business on Open Source technology.

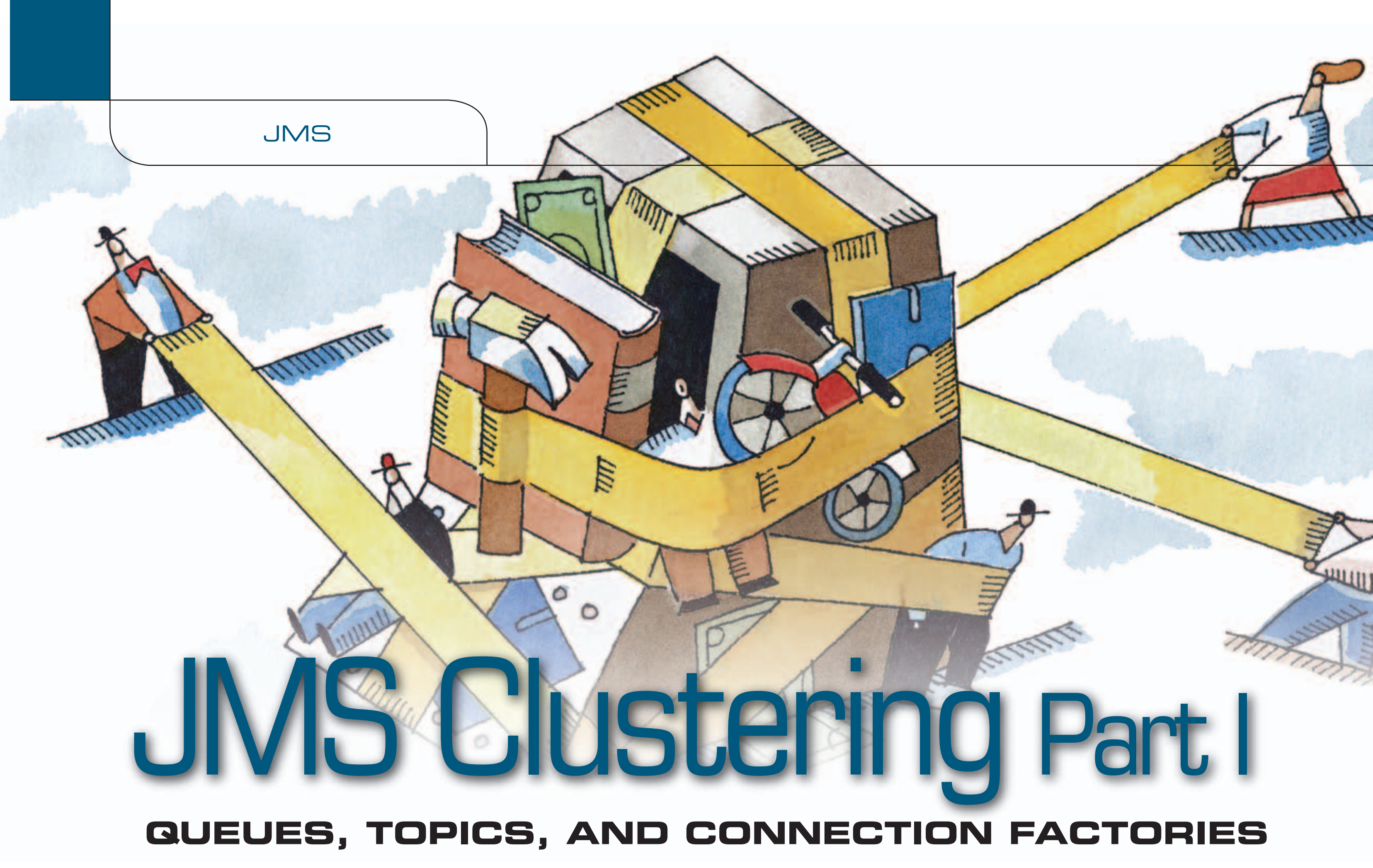
NETWORK with your peers and the be first to meet the hottest up and coming companies in the Open Source industry.

CORNERSTONE SPONSOR

Sun microsystems

CORPORATE SPONSORS

intel. Microsoft Novell. ca Computer Associates OSDL REALM optaros



JMS Clustering Part I

QUEUES, TOPICS, AND CONNECTION FACTORIES

By Raffi Basmajian

This article is the first of a two-part series on JMS clustering. Part 1 will discuss the fundamental aspects of clustering JMS resources such as queues, topics, and connection factories, and illustrate the steps to go through to configure a clustered destination in a WebLogic cluster. Part 2 will discuss JMS clustering in the context of several design and configuration strategies that demonstrate how to create efficient and optimized JMS architectures.

WebLogic v7.0 introduced JMS clustering. This article will discuss the fundamental aspects of JMS clustering in WebLogic 8.1 (SP3).

Clustering JMS Resources

Scalability, high availability, and fault tolerance are required of mission-critical systems. The standards that have been memorialized in J2EE provide the framework needed to build the robust architecture that meets those requirements. JMS provides enterprise systems with a foundation for messaging. JMS systems that demand near-zero downtime, such as those in financial houses, must provide the high availability and failover for JMS resources to meet stringent service-level agreements. So it's vital to use a JMS implementation that lets JMS resources, such as queues, topics, and connection factories, be clustered and highly available while load-balancing message traffic among distributed JMS destinations.

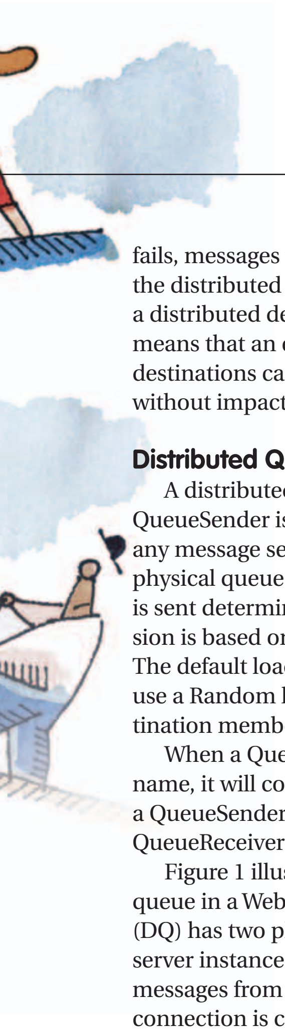
WebLogic's JMS is a complete messaging implementation that supports JMS standards. It also provides services so that clustered JMS configurations can load-balance messages and offer high availability for all kinds of JMS resources. In a WebLogic cluster, ordinary queues and topics can be configured as distributed destinations that consist of many physical destinations dispersed among physical machines, creating a highly available messaging layer. Connection factories can also be distributed. It lets WebLogic load-balance connection requests from JMS clients and transparently directs message traffic to the desired destination, or redirects messages in the event of destination failure.

Distributed Destinations

A distributed destination represents a group of physical queues, or topics, whose members are hosted by JMS servers in a WebLogic cluster. The distributed destination is bound to a logical JNDI name in the cluster-wide JNDI tree. WebLogic will load-balance messages among the members of a distributed destination and, if a destination member

Author Bio:
Raffi Basmajian is an application architect for a major financial services firm in New York City. He specializes in architecting Java/J2EE distributed systems and integration frameworks.

Contact:
raffibasmajian@yahoo.com



fails, messages will be transparently redirected to other members in the distributed destination. From the point-of-view of a JMS client, a distributed destination appears as an ordinary destination, which means that an existing JMS application configured with ordinary destinations can change its configuration to distributed destinations without impacting JMS clients or application code.

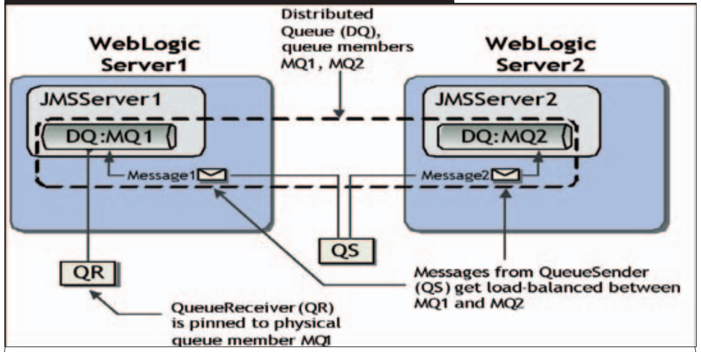
Distributed Queues

A distributed queue represents a group of physical queues. If a QueueSender is created using the JNDI name of a distributed queue, any message sent from that QueueSender is delivered to only one physical queue destination. A decision is made every time a message is sent determining which member will get the message. This decision is based on a load-balancing algorithm provided by WebLogic. The default load-balancing scheme is a Round Robin, but you can use a Random load-balancing scheme to distribute messages to destination members at random.

When a QueueReceiver is created using a distributed queue name, it will connect to a physical queue member and, unlike a QueueSender, remain pinned to that destination until the QueueReceiver loses connection.

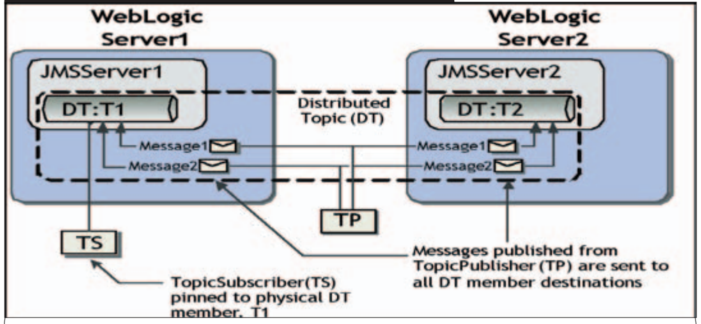
Figure 1 illustrates how JMS clients interact with a distributed queue in a WebLogic cluster. In the diagram, the distributed queue (DQ) has two physical queue members MQ1 and MQ2 residing on server instances 1 and 2, respectively. The queue receiver (QR) gets messages from MQ1 and will remained pinned to MQ1 until the connection is closed. The queue sender (QR) sends messages to the distributed queue DQ. Only one physical queue member gets the messages sent to a distributed queue and, as the illustration shows,

FIGURE 1


WebLogic Server1
Distributed Queue (DQ), queue members MQ1, MQ2
WebLogic Server2
JMSServer1
DQ:MQ1
JMSServer2
DQ:MQ2
Message1
Message2
QS
QR
Messages from QueueSender (QS) get load-balanced between MQ1 and MQ2
QueueReceiver (QR) is pinned to physical queue member MQ1

A QueueSender and QueueReceiver interacting with a distributed queue destination

FIGURE 2


WebLogic Server1
WebLogic Server2
JMSServer1
DT:T1
JMSServer2
DT:T2
Distributed Topic (DT)
Message1
Message2
Message1
Message2
TP
TS
Messages published from TopicPublisher (TP) are sent to all DT member destinations
TopicSubscriber(TS) pinned to physical DT member. T1

A TopicPublisher and TopicSubscriber interacting with a distributed topic destination

the first message, Message1, sent by QS is load-balanced and sent to MQ1. The next message, Message2, is also load-balanced and is sent to the next queue member in the sequence, MQ2. The illustration uses a Round Robin load-balancing policy, which picks server instances in an ordered sequence.

When a message is sent to a distributed queue member with no consumers, the message will remain in that queue until a consumer is available, and, if the queue member fails, all messages pending in that queue will be unavailable until the queue member is available again. Configuring a Forward Delay on each member in the distributed queue can prevent this situation. This option automatically forwards messages from a queue member with no active consumers to queue members with active consumers. By default, the Forward Delay is disabled. We'll discuss how to configure this destination parameter below.

If a physical queue member fails, all consumers connected to it are notified by a JMSException and effectively lose their connection to the queue. For synchronous consumers, the exception is returned directly to the client. For asynchronous consumers whose queue session is configured with an ExceptionListener, a ConsumerClosedException is sent. In either case – assuming the failure was isolated to the physical queue member – the JMS connection and session remain valid. The client application just closes the queue receiver and recreates it using the same JMS session.

Distributed Topics

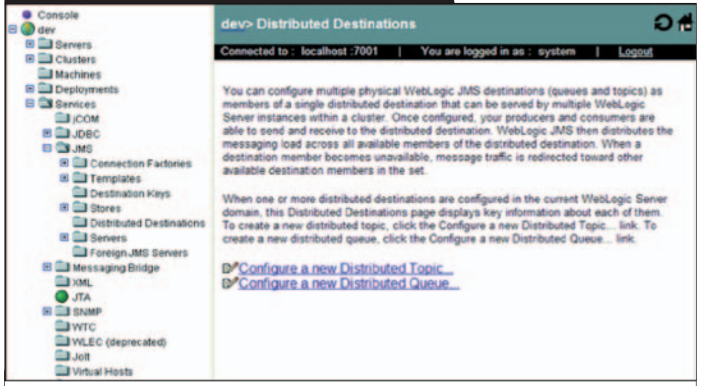
A distributed topic represents a group of physical topics. JMS client applications can create topic message producers and consumers using a distributed topic name. Your application doesn't know how many physical destination members the distributed topic has, which eliminates any special programming considerations when using this kind of clustered JMS resource.

When the TopicPublisher created with a distributed topic name publishes a message, the message is automatically sent to all members of the distributed topic so all subscribers get the message. Messages are forwarded to all topic member destinations even if a TopicPublisher is created using the JNDI name of a physical topic destination and not the distributed topic JNDI name.

Non-persistent messages published to a distributed topic are sent only to the topic destinations that are available. If your application uses persistent messages, then it's a good idea to configure each topic member destination with a persistent store. WebLogic gives preference to topic destinations configured with persistent stores when publishing a persistent message on a distributed topic to avoid lost messages. If any topic members are unavailable when a persistent message is published, the message will be stored on the topic members with persistent stores and forwarded to the remaining topic members as they become available.

A TopicSubscriber created with a distributed topic name is pinned to a physical topic member of the distributed topic. Connection is maintained until the destination topic member fails. If and when that happens, the topic subscriber is sent a JMSException the same as when a destination queue member fails. If a topic destination member with active subscribers fails, then any persistent messages published to the distributed topic after the failure won't be delivered to the disconnected subscribers until the topic destination member can get messages again. In contrast, a subscriber that lost

FIGURE 3



dev> Distributed Destinations

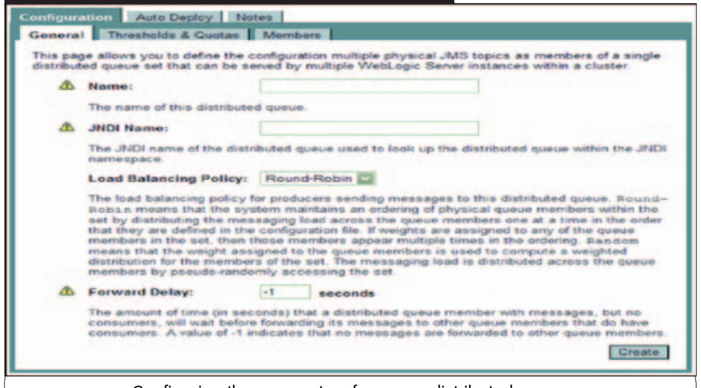
Connected to : localhost :7001 | You are logged in as : system | Logout

You can configure multiple physical WebLogic JMS destinations (queues and topics) as members of a single distributed destination that can be served by multiple WebLogic Server instances within a cluster. Once configured, your producers and consumers are able to send and receive to the distributed destination. WebLogic JMS then distributes the messaging load across all available members of the distributed destination. When a destination member becomes unavailable, message traffic is redirected toward other available destination members in the set.

When one or more distributed destinations are configured in the current WebLogic Server domain, this Distributed Destinations page displays key information about each of them. To create a new distributed topic, click the Configure a new Distributed Topic... link. To create a new distributed queue, click the Configure a new Distributed Queue... link.

Configure a distributed queue or topic from the Administration Console

FIGURE 4



This page allows you to define the configuration multiple physical JMS topics as members of a single distributed queue set that can be served by multiple WebLogic Server instances within a cluster.

Name:

The name of this distributed queue.

JNDI Name:

The JNDI name of the distributed queue used to look up the distributed queue within the JNDI namespace.

Load Balancing Policy: Round-Robin

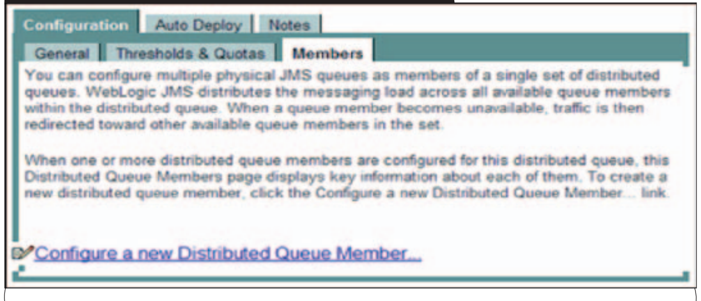
The load balancing policy for producers sending messages to this distributed queue. Round-Robin means that the system maintains an ordering of physical queue members within the set by distributing the messaging load across the queue members one at a time in the order that they are defined in the configuration file. If weights are assigned to any of the queue members in the set, then those members appear multiple times in the ordering. Random means that the weight assigned to the queue members is used to compute a weighted distribution for the members of the set. The messaging load is distributed across the queue members by pseudo-randomly accessing the set.

Forward Delay: -1 seconds

The amount of time (in seconds) that a distributed queue member with messages, but no consumers, will wait before forwarding its messages to other queue members that do have consumers. A value of -1 indicates that no messages are forwarded to other queue members.

Configuring the parameters for a new distributed queue

FIGURE 5

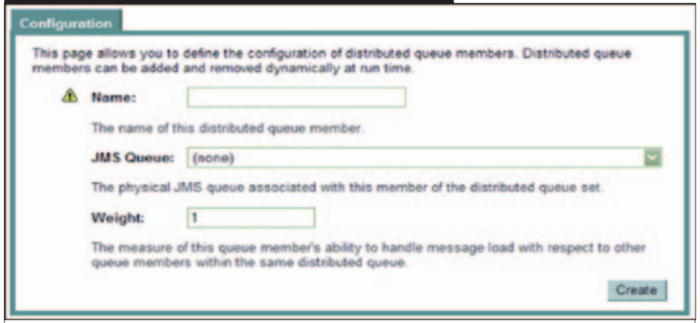


You can configure multiple physical JMS queues as members of a single set of distributed queues. WebLogic JMS distributes the messaging load across all available queue members within the distributed queue. When a queue member becomes unavailable, traffic is then redirected toward other available queue members in the set.

When one or more distributed queue members are configured for this distributed queue, this Distributed Queue Members page displays key information about each of them. To create a new distributed queue member, click the Configure a new Distributed Queue Member... link.

Configure physical queue members for a distributed queue

FIGURE 6



This page allows you to define the configuration of distributed queue members. Distributed queue members can be added and removed dynamically at run time.

Name:

The name of this distributed queue member.

JMS Queue: (none)

The physical JMS queue associated with this member of the distributed queue set.

Weight: 1

The measure of this queue member's ability to handle message load with respect to other queue members within the same distributed queue.

Set the parameters for a distributed queue member

connection with a distributed topic member never gets a non-persistent message.

Figure 2 illustrates how JMS clients interact with a distributed topic in a WebLogic cluster. The distributed topic (DT) in the diagram consists of two physical topic members T1 and T2 residing on server instances 1 and 2, respectively. A topic subscriber (TS) gets messages from T1 and remains pinned to this topic member until the connection is closed or lost. A topic publisher (TP) sends messages to the distributed topic (DT). All physical topic members get the messages sent to the distributed topic. As the illustration shows, TP publishes Message1 to the distributed topic DT, which sends it to topic members T1 and T2. The next message Message2 is also sent to both topic members.

Clustering Connection Factories

Connection factories can also leverage the capabilities of JMS clustering to provide load balancing, high availability, and failover for JMS clients.

A connection factory can be targeted to a cluster, or targeted to individual WebLogic instances in a cluster, whether or not those WebLogic instances host a JMS server. From an architectural perspective, you need to consider the physical location of the JMS clients in your application and the destinations they will access before you decide where to target your connection factories.

For example, if an external client uses a connection factory targeted to multiple WebLogic instances in a cluster, the decision of which connection factory to use as well as which JMS server will host the JMS connection is load-balanced.

The connection routing might be inefficient if the new JMS connection and the local JMS server are on the same physical machine. To avoid unnecessary connection routing, make sure the *Server Affinity* parameter for each connection factory in your JMS application is enabled. You can change this setting from the WebLogic Administration console by navigating to the "Services/JMS/Connection Factories" node. Select the connection factory from the list, and scroll to the bottom of the *General* tab to find the *Server Affinity* parameter.

In contrast, an internal JMS client trying to locate a connection factory and create a JMS connection won't incur a wasteful remote connection if the connection factory is physically located on the same WebLogic instance as the JMS server that hosts the desired destination. The default load-balancing policy used by the connection factory for internal JMS clients is circumvented because WebLogic gives preference to collocated connection factories and JMS servers to avoid creating remote connections.

The details of configuring and targeting connection factories in the context of different JMS application scenarios will be discussed in part two of this article.

Configuring a Distributed Destination

To configure a distributed queue or topic, open the WebLogic Administration console and navigate to the "Services/JMS/Distributed Destinations" node (Figure 3).

Now you can create a Distributed Topic or a Distributed Queue. In this example we'll create a distributed queue. To continue, click *Configure a new Distributed Queue*. Figure 4 displays the configuration parameters for creating a distributed queue.

Name: The unique logical identifier for the distributed queue destination.

THE WORLD'S LEADING INDEPENDENT WEBLOGIC DEVELOPER RESOURCE

ARE YOU A WEBLOGIC EXPERT? PAGE 4
NOV/DEC 2004
wldj
THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC PROFESSIONALS
WWW.WLDJ.COM
Database Controls
Best Practices
AVOIDING POTENTIAL PITFALLS DURING THE DEVELOPMENT LIFE CYCLE ...8
PLUS...
Open Source Technologies ...6
Enabling Next-Generation Portals ...14
Security Best Practices ...18
Strategies for WebLogic Domain Configuration ...20
Creating WebLogic Domains in Silent Mode ...26
A Real-World Business Process Model PART 3 ...36
TRANSACTIONS: And Now for Something Completely Different page 32
PRODUCT REVIEW: SOAPtest page 40
PORTAL: Enterprise Information Bus page 46
FREE Web Services
Web Services Journal Pavilion
Seminars and Case Studies

Helping you enable intercompany collaboration on a global scale

- Product Reviews
- Case Studies
- Tips, Tricks and More!

SAVE $69.89 OFF REGULAR NEWSSTAND PRICE

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

*Only $49.99 for 2 years (12 issues)

Now in More Than 5,000 Bookstores Worldwide

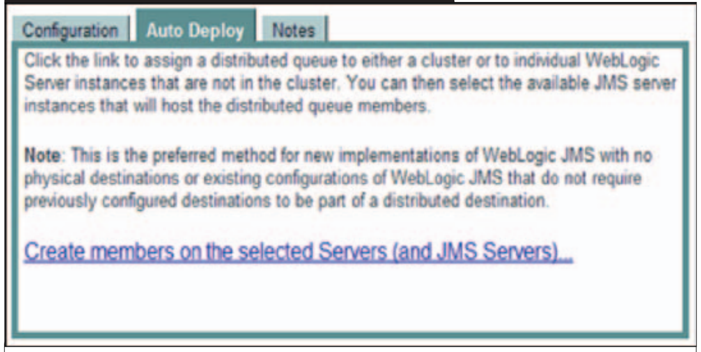
Go Online & Subscribe Today! www.WLDJ.com

SYS-CON.COM/WEBLOGIC/

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of WebLogic.

SYS-CON MEDIA

FIGURE 7



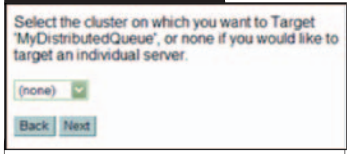
Click the link to assign a distributed queue to either a cluster or to individual WebLogic Server instances that are not in the cluster. You can then select the available JMS server instances that will host the distributed queue members.

Note: This is the preferred method for new implementations of WebLogic JMS with no physical destinations or existing configurations of WebLogic JMS that do not require previously configured destinations to be part of a distributed destination.

Create members on the selected Servers (and JMS Servers)...

Auto Deploy creates physical destination members automatically for the distributed destination

FIGURE 8

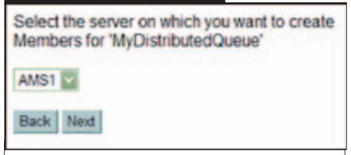


Select the cluster on which you want to Target 'MyDistributedQueue', or none if you would like to target an individual server.

(none)

Select a cluster or individual managed servers to serve as the target for the distributed destination

FIGURE 9

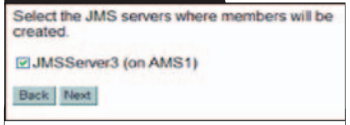


Select the server on which you want to create Members for 'MyDistributedQueue'

AMS1

Select the WebLogic server for new distributed destination members

FIGURE 10

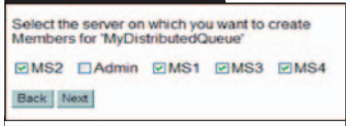


Select the JMS servers where members will be created.

[✓] JMSServer3 (on AMS1)

Select the JMS servers for hosting destination members

FIGURE 11



Select the server on which you want to create Members for 'MyDistributedQueue'

[✓] MS2 [ ] Admin [✓] MS1 [✓] MS3 [✓] MS4

Select the cluster servers for new distributed destination members

JNDI Name: The lookup alias for cluster-wide JNDI.
Load Balancing Policy: The algorithm used to determine the distribution of messages among the distributed destination members.
Forward Delay: Only for distributed queues. It defines the amount of time a distributed queue member with no consumers will wait before forwarding its pending messages to other queue members with active consumers.

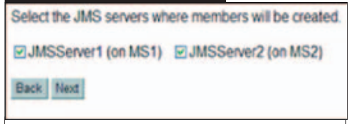
Once you create the distributed queue click the *Thresholds and Quotas* tab. The parameters on this screen are well documented so we'll skip the point of each setting, but in general the parameters let you configure the maximum message quotas, message thresholds, maximum message size, and bytes message paging. These settings apply globally to all physical members belonging to the distributed destination.

At this point, you're ready to add physical queue members to the distributed queue. If your WebLogic domain has queue destinations that you want to designate as members of your newly created distributed queue, click the Members tab (Figure 5), then click *Configure a new Distributed Queue Member*.

On the next screen, enter the configuration parameters of the physical queue member (Figure 6).

Name: The unique logical identifier for the member destination.
JMS Queue/Topic: The list of available queues from which you select the physical destination of new members of the distributed queue.

FIGURE 12



Select the JMS servers where members will be created.

[✓] JMSServer1 (on MS1) [✓] JMSServer2 (on MS2)

Select the JMS servers for hosting destination members

Weight: The integer that controls the message load balancing for the Round Robin algorithm. The weights are relative to the other queue member destinations in the same distributed queue; a higher value designates the members that get more messages than those with a lower setting.

Click the Create button to create the new distributed queue member. You can repeat this procedure for adding other members to the distributed queue.

WebLogic provides a convenient feature to create physical queues or topics automatically and assign them to a distributed destination.

To use this feature, click the Auto Deploy tab (Figure 7) and then click *Create members on the selected Servers (and JMS Servers)*.

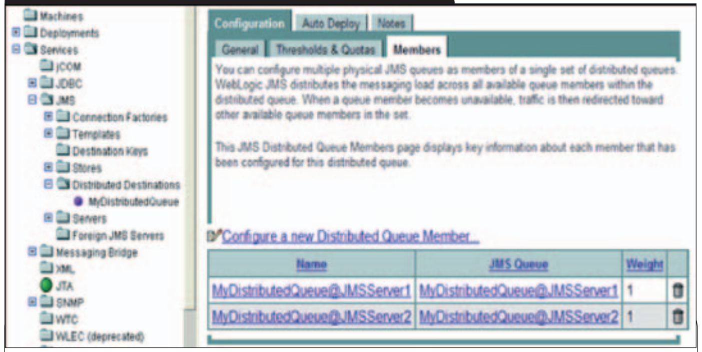
On the next screen you have the option of choosing the WebLogic instances directly, or the WebLogic cluster, to serve as the targets for the distributed destination.

If you want to target a WebLogic instance directly:
Select *(none)*, then click *Next* (Figure 8)
Select the WebLogic instance on which to create destination members, then click *Next* (Figure 9)
Select the JMS servers on which to deploy the destinations members, then click *Next* (Figure 10)

If you want to target to a WebLogic cluster:
Select the cluster name, then click *Next* (Figure 8)
Select the WebLogic instances on which to create destination members, then click *Next* (Figure 11)
Select the JMS servers on which to deploy the destinations members, then click *Next* (Figure 12)

After applying the configuration changes, WebLogic automatically creates member destinations on the JMS servers selected. Click the Members tab to see the list of newly generated member destinations (Figure 13).

FIGURE 13



You can configure multiple physical JMS queues as members of a single set of distributed queues. WebLogic JMS distributes the messaging load across all available queue members within the distributed queue. When a queue member becomes unavailable, traffic is then redirected toward other available queue members in the set.

This JMS Distributed Queue Members page displays key information about each member that has been configured for this distributed queue.

Configure a new Distributed Queue Member...

Name	JMS Queue	Weight	
MyDistributedQueue@JMSServer1	MyDistributedQueue@JMSServer1	1	🗑
MyDistributedQueue@JMSServer2	MyDistributedQueue@JMSServer2	1	🗑

The newly created destination queue members using Auto Deploy

Conclusion

At this point you should understand the basic features of JMS clustering and the kinds of distributed resources it provides. The next article will discuss JMS application design strategies for internal producers/consumers, external-only producers, highly distributed and load-balanced JMS applications, and the best practices and guidelines to follow when using JMS clustering.

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Acsera	www.acsera.com/register	408-245-1000	51
EV1 Servers	www.ev1servers.net	800-504-SURF	13
H&W Computer	www.hwcs.com	800-338-6692	17
HP	http://devresource.hp.com	800-752-0900	23
Information Storage & Security	www.issjournal.com	888-303-5282	41
Intersperse	www.intersperse.com	866-499-2202	2
IT Solutions Guide	www.sys-con.com	201-802-3021	41
LinuxWorld Magazine	www.linuxworld.com	888-303-5282	35
Motive	www.motive.com/within1	512-531-2527	52
NetIQ	www.netiq.com/solutions/web	408-856-3000	25
OSBC	www.osbc.com	801-358-4231	29
Pantero	www.pantero.com	781-890-2890	3
Parasoft	www.parasoft.com/soaptest_wdj	626-256-3680	9
Quest Software	www.quest.com/wldj	949-754-8633	7
SandCherry	www.sandcherry.com	866-383-4500	27
SYS-CON e-newsletters	www.sys-con.com	888-303-5282	35
SYS-CON Publications	www.sys-con.com/2001/sub.cfm	888-303-5282	43
SYS-CON Reprints	www.sys-con.com	201-802-3026	45
Tangosol	www.tangosol.com	617-623-5782	15
Wily Technology	www.wilytech.com	888-GET-WILY	5
WLDJ	www.sys-con.com/weblogic/	888-303-5282	33

This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions

Monitoring and Controlling WebLogic Servers with WLST

WRITING SCRIPTS MERGED WITH JAVA IS MORE NATURAL THAN ANT

By Ignacio Coloma

Scripting languages have recently garnered a bit of attention. With the arrival of Groovy and Jython, writing scripts merged with Java is more natural than Ant. Using XML to call Java methods has always been forced, mostly because it's hard to express flow, conditional expressions, and custom Java code in a markup language (although things have improved a lot since Ant 1.5).

Why a scripting language? Well, if I have a completely blown IDE for Java programming, using Jython or Groovy can look backwards. You can code in fewer lines (though not much less), but I want the imports written automatically. I want compiler warnings while coding and I need refactorings. Plug-ins for these languages are still outside of Java IDE's capabilities.

But there are times where you just don't have a full IDE configured. Think about jobs that should be automated to be agile, or about server administrators. These people don't have anything like Eclipse, and their work can't be done in advance. You can't code for system administration. This is where tools like WLST come in and make the world a better place.

WLST (WebLogic Scripting Tool) is a Jython module that helps write scripts to administer and modify a server installation remotely. It comes in two flavors: offline, which can configure a server instance that doesn't exist yet, and online, which needs a WebLogic server to connect to. Both versions are in beta and are poorly documented, but they promise to improve and will be in some future WebLogic release.

We're going to focus on the online version here, because its functionalities are more complete than the offline version.

Automating Server Configuration:

Let's start by getting rid of that nasty WebLogic server configuration. Your typical development team replicates the same config in several hosts, changing only a couple of parameters such as the IP, hostname, and TCP port. In a relatively typical project, the process must be executed for each developer's PC, integration test host, and production. Ant tasks resolve great in this context, but it's not prepared to handle things like custom JMX beans.

We're going to create and launch the server, configure it, and do a shutdown, using a mixture of ant and WLST. First, let's create the server in Listing 1. For simplicity's sake, we're going to use the ant task here because combining WLST offline and online would mess things up.

I check the properties because when you deploy on more than one brand of app server it's easy to use the wrong build.properties file (see Listing 2).

We have just removed the whole domain directory, created a new clean one, and left the server running, so now in Listing 3 we can connect and configure it.

Author Bio:
Ignacio Coloma is a J2EE architect at InfoInnova. For the last six years he's been developing applications for e-banking, air transport, e-government, and message processing systems. He is currently extending J2EE applications with scripting languages.

Contact:
icoloma@infoinnova.com

The server stop is necessary because some setting changes, i.e., security authenticators, need a graceful shutdown to be stored on disk. Omitting this step would kill the server in a hard way at the end of the ant script

Note: the WLST task is forked, and so, if WLST finds an error in your script, ant still will say "build successful," something that might confuse the person launching the script.

Let's split the WLST script into two parts to reuse as much of it as possible for administration tasks later. I have used the great examples enclosed with the WLST bundle and the output of the saveDomain() command as starting points. The saveDomain() generated script isn't very polished, but it serves to indicate the tool's possibilities (see Listing 4).

The loadProperties task converts all the entries in the administration.properties file to Jython variables. We've used the first methods of a Jython class to administrate the WebLogic server instance. It can easily be extended to create and remove DataSources, a JMS environment, and even security realms.

MBean Methods

What you have seen is a way of creating and configuring MBeans (there's another way that will be explained in the next section). The downside is that you have to know the attributes and methods supported, and WLST doesn't document them. How can I guess which methods are available?

Well, the first way that comes to mind is going to the config.xml file or the web console and assume the attribute names hasn't changed. If we have a decent IDE we also can open the MBean interface class and see what's in there (it's the same as the MBean name, ending with 'MBean'). It won't show you code, but you can check which methods are available.

I prefer connecting to http://e-docs.bea.com/wls/docs81/javadocs/index.html and checking the contents of the package weblogic.management.configuration. For example, if we go to the ServerMBean class, we can see two interesting and not particularly well-known methods isJDBCLoggingEnabled() and setJDBCLoggingEnabled(). We can check them by opening the wlst interactive shell as laid out below:

```
wls:/mydomain/config> server=home.getAdminMBean('myserver', 'Server')
wls:/mydomain/config> server.setJDBCLoggingEnabled(1)
wls:/mydomain/config> server.isJDBCLoggingEnabled()
1
```

('home' is a variable of type AdminMbeanHomeImpl and can be studied as any other MBean; the only problem is that there's no javadoc available since it's an internal class).

If these last three commands aren't easily understood, don't worry. The shell will be covered in the next section.

Command Line System Administration

A system administrator can also administer the WebLogic server instance manually by using the interactive shell. The advantage here is you don't have to know a priori the MBean interfaces when trying to modify the system config. For this part, you have to include weblogic.jar, jython.jar, and wist.jar in the classpath and start the main class weblogic.WLST, which is the interactive console.

Keep it mind that this is Jython. Quotes and double quotes are used for string delimitation; instantiation doesn't need a *new* operator (as a matter of fact it's a syntax error); semicolons aren't required because each line ends with a carriage return; and variables don't have to be declared (a la Unix shell scripts). If that's not enough for you, please refer to the Python and WLST docs.

We need to start connecting to a WebLogic server instance. We can choose to use the AdminTool script developed before, or connect manually:

```
execfile('AdminTool.py')
admin.connect()
```

or

```
connect('weblogic', 'weblogic', «t3://localhost:7001)

Connecting to weblogic server instance running at t3://127.0.0.1:7001 as
username weblogic ...
```

Successfully connected to Admin Server 'myserver' that belongs to domain 'mydomain' is system output and should be formatted as code.

Now we can start playing with it. With WLST, the JMX tree can be traversed as a Unix filesystem, where the JMX MBeans are directories and its attributes are files. Keep in mind the Python syntax all the way through, and remember that WLST still doesn't recognize wildcards. That's the reason why we're going to omit most of the ls() output (see Listing 5).

We could have also gotten that far on a single cd('/JDBCConnectionPools/MyPool') command. WLST always remembers the cmo (Current Managed Object), the MBean corresponding to the current 'folder' we're browsing. So, these commands are equivalent from the practical point-of-view:

```
wls:/mydomain/config/JDBCConnectionPools/MyPool> cmo
[Caching Stub]Proxy for mydomain:Name=MyPool,Type=JDBCConnectionPool
wls:/mydomain/config/JDBCConnectionPools/MyPool> pwd()
'/JDBCConnectionPools/MyPool'
```

Now, let's change a couple of random properties (see Listing 6). Remember that Python doesn't have boolean attributes. The server can return true and false (since it runs Java), but you can't assign those values. Don't worry, though; if you check it through the WebLogic console, your boolean value of 1 has been interpreted correctly by the server.

You could have gotten the same result using the equivalent techniques shown in the section above about "Automating Server Configuration." I find this way easier for systems administrators, and the first way for developers preparing scripts. It just fits better in each different kind of toolset: system administrators are more used to Unix shells, and developers feel more comfortable with the "smell" of Java.

Managing the Server Configuration Example: A Real Case

It's common to need to peep inside those nasty JDBC calls. One sometimes really wants to be able to see the conversation between the WebLogic server and the database, and why in hell the query returns 0 rows, or profile performance, okay. Logging the JDBC calls (not just the SQL, please, but the parameters too) with a hot-plug capability should be nice. Wanna give it a try?

First, let's download the p6spy JDBC driver. It is a JDBC wrapper that will log anything that goes through it. To configure it, put the p6spy.jar and the directory containing the p6spy.properties in the server classpath (don't forget the directory, or WebLogic will complain as if the JAR file were not present). Tune the p6spy.properties to your needs.

What we want to achieve is the creation of two Connection Pools, one directly with Oracle JDBC driver and the other through p6spy. Then, we will change the datasource to point to the p6spy datasource without restarting the server (if we believe the Web console interface, this change does not need a reboot).

We will start by executing the administration script developed earlier:

```
wls:/(offline)> execfile('AdminTool.py')

Connecting to weblogic server instance running at t3://127.0.0.1:7001 as
username weblogic ...
```

Successfully connected to Admin Server 'myserver' that belongs to domain 'mydomain'. It also is system output and should be formatted accordingly.

We can create the Connection Pool now.

```
wls:/mydomain/config> admin.createPool("P6SPY Connection Pool", "com.
p6spy.engine.spy.P6SpyDriver")
JDBCConnectionPool with name 'P6SPY Connection Pool' has been created
successfully.
```

In the WebLogic console we can see the following (well, the WebLogic log line will only appear if you have the 'debug to console' option activated):

```
<28-feb-2005 20H18' GMT> <Info> <JDBC> <BEA-001132> <Initialized state-
ment cache of size "10" for co
nnection in pool "P6SPY Connection Pool".>
1109621928226|0|1|statement|SELECT 1 FROM DUAL|SELECT 1 FROM DUAL
1109621928242|0|1|statement|SELECT 1 FROM DUAL|SELECT 1 FROM DUAL
```

which shows the connection pool initializing and new connections test. We'll suppose that the dataSource doesn't exist yet. If we were clean, we would have foreseen it and created the method in our AdminTool class, but we can still do it via the interactive shell in Listing 7.

We have started directing the datasource to the P6SPY connection pool, so you can check your application and see that it really logs JDBC statements; try it with a test case. Now, there are two ways to disable the logging. Since we have the datasource in a Jython variable, we can do it the 'Java' way:

```
datasource.setPoolName(MY_POOL_NAME)
```

or, the 'system administrator' way shown in Listing 8.

Conclusion

WLST is an awesome tool capable of boosting your application server configuration and remote maintenance. It still lacks a find/locate option (for the not uncommon case where one needs to find a configuration option and can't recall its location) with wildcard support. But when it finally gets bundled with WebLogic 9 it's sure to be useful.

Resources

- *The p6spy open source driver:* www.p6spy.com/
- *Download WLST online:* http://dev2dev.bea.com/codelibrary/code/wlst.jsp
- *Download WLST offline:* http://dev2dev.bea.com/codelibrary/code/wlst_offline.jsp
- *Strategies for WebLogic domain configuration:* www.sys-con.com/story/?storyid=47096&DE=1
- *Martin Fowler on scripting languages for complex tasks not easily achieved with XML:* http://martinfowler.com/bliki/BuildLanguage.html

Listing 1

```
build.properties:
server=weblogic
weblogic.server.name=myserver
weblogic.domain.name=mydomain
weblogic.admin.username=weblogic
weblogic.admin.password=weblogic

weblogic.home=C:/bea
weblogic.lib.dir=${weblogic.home}/weblogic81/server/lib
weblogic.mbeantypes.dir=${weblogic.lib.dir}/mbeantypes
server.project.root.dir=${weblogic.home}/user_projects/domains/\
${weblogic.domain.name}
server.deploy.dir=${server.project.root.dir}/applications
```

Listing 2

```
build.xml:
```

```
[...]

<path id="weblogic.classpath">
        <fileset dir="${weblogic.lib.dir}">
            <include name="weblogic.jar"/>
        </fileset>
</path>

<target name="check-properties">
    <condition property="wlproperties.ok">
    <and>
          <isset property="weblogic.server.name"/>
          <isset property="weblogic.domain.name"/>
          <isset property="weblogic.admin.username"/>
          <isset property="weblogic.admin.password"/>
    </and>
    </condition>
    <fail unless="wlproperties.ok">
```

```
     Weblogic properties are missing. Double check build.properties.
   </fail>

     <fail>
         <condition>
           <not><available file=»${weblogic.lib.dir}/weblogic.jar»/></
             not>
         </condition>
     Missing file ${weblogic.lib.dir}/weblogic.jar
     </fail>
</target>


<target name="create-server" depends="check-properties">

     <taskdef name="wlserver" classname="weblogic.ant.taskdefs.
       management.WLServer"
        classpathref="weblogic.classpath"
     />

     <echo>Creating server ${weblogic.server.name} at ${server.project.
       root.dir}</echo>
     <delete dir="${server.project.root.dir}" includeemptydirs="true"
       quiet="true"/>
     <mkdir dir="${server.project.root.dir}" />
     <wlserver
        dir="${server.project.root.dir}"
        domainName="${weblogic.domain.name}"
        serverName="${weblogic.server.name}"
        host="${host.ip}"
        port="${rmi.port}"
        generateConfig=»true»
        productionModeEnabled=»false»
        weblogicHome=»${weblogic.home}/weblogic81»
          username=»${weblogic.admin.username}»
        password=»${weblogic.admin.password}»
        action=»start»
        >
        <classpath refid=»weblogic.classpath»/>
     </wlserver>
```

Listing 3

```
<java classname="weblogic.WLST" fork="true" failOnError="true"
dir="scripts/wlst">
        <classpath refid="weblogic.classpath"/>
        <classpath>
          <fileset dir="lib/wlst">
              <include name="*.jar"/>
          </fileset>
        </classpath>
          <arg line="createAll.py" />
     </java>

     <wlserver
        host="${host.ip}"
        port="${rmi.port}"
          username="${weblogic.admin.username}"
          password="${weblogic.admin.password}"
          action="shutdown"
        />

</target>
```

Listing 4

```
createAll.py:
execfile("AdminTool.py")
admin.configureServer()
admin.createXaPool()


AdminTool.py:
from javax.management import InstanceNotFoundException

# Python 2.4 will include booleans, but until then this is required
true = 1
false = 0

class AdminTool:

  def __init__(self):
      loadProperties("administration.properties")

  # Connects with a weblogic instance
  def connect(self):
      connect(username, password, "t3://" + host + ":" + port)
      self.myServer = getTarget("/Server/" + serverName)

  # Server attributes that cannot be generated via ant
  def configureServer(self):
      # Activates console DEBUG mode - I really like that
      self.myServer.setStdoutSeverityLevel(64)
      print "Configured server " + self.myServer.getName()

  # Creates a JDBC pool:
  def createPool(self, poolName, driverName):
      pool = create(poolName, "JDBCConnectionPool")
      pool.setDriverName(driverName)
      pool.setURL(dbURL)
      pool.setPassword(dbPassword)
      pool.setProperties(makePropertiesObject("user=" + dbUsername))
      pool.setRemoveInfectedConnectionsEnabled(false)
      pool.setTestConnectionsOnCreate(true)
      pool.setTestTableName("SQL SELECT 1 FROM DUAL")
      # setTestFrecuencySeconds is not soported by WLST objects
      # so here is a workaround
      cd('/JDBCConnectionPool/' + poolName)
      set('TestFrequencySeconds', 300)
      cd('/')
      pool.addTarget(self.myServer)

  def createXaPool(self):
      self.createPool(MY_POOL_NAME, 'oracle.jdbc.xa.client.
       OracleXADataSource')
```

```
    # Removes an element if it exists
    def removeIfExists(self, name, type):
        try:
            mbean = home.getAdminMBean(name, type)
            home.deleteMBean(mbean)
            print 'Removed the ' + type + ': ' + name
        except InstanceNotFoundException, e:
            print "Cannot remove " + name + ",type=" + type + " because
             it does not exist"

    def removeXaPool(self):
        self.removeIfExists(MY_POOL_NAME, "JDBCConnectionPool")

# The JDBC Connection Pool name
MY_POOL_NAME='MyPool'

# the instance we are going to use
admin = AdminTool()
admin.connect()


administration.properties:
host=127.0.0.1
port=7001
username=weblogic
password=weblogic

dbURL=jdbc:oracle:thin:@dbhost:1521:DATABASE
dbUsername=foo
dbPassword=bar
```

Listing 5

```
wls:/mydomain/config> ls()
[...]
drw-   JDBCConnectionPools
drw-   JDBCDataSourceFactories
drw-   JDBCDataSources
drw-   JDBCMultiPools
drw-   JDBCTxDataSources
[...]

wls:/mydomain/config> cd('JDBCConnectionPools')

wls:/mydomain/config/JDBCConnectionPools> ls()
drw-   MyPool

wls:/mydomain/config/JDBCConnectionPools> cd('MyPool')

wls:/mydomain/config/JDBCConnectionPools/MyPool> ls()
[...]
-rw-   TestConnectionsOnCreate                    true
-rw-   TestConnectionsOnRelease                   false
-rw-   TestConnectionsOnReserve                   false
-rw-   TestFrequencySeconds                       300
-rw-   TestStatementTimeout                       -1
-rw-   TestTableName                              SQL SELECT 1 FROM DUAL
-r--   Type                                       JDBCConnectionPool
-rw-   URL                                        jdbc:oracle:
thin:@dbhost:1521:DATABASE
[...]
```

Listing 6

```
wls:/mydomain/config/JDBCConnectionPools/MyPool> set
  ('TestFrequencySeconds', 500)
wls:/mydomain/config/JDBCConnectionPools/MyPool> set('TestConnectionsOn
Release', 1)
wls:/mydomain/config/JDBCConnectionPools/MyPool> ls()
-rw-   TestConnectionsOnCreate                true
-rw-   TestConnectionsOnRelease               1
-rw-   TestConnectionsOnReserve               false
-rw-   TestFrequencySeconds                   500
-rw-   TestStatementTimeout                   -1
-rw-   TestTableName                          SQL SELECT 1 FROM DUAL
-r--   Type                                   JDBCConnectionPool
-rw-   URL                                    jdbc:oracle:
    thin:@dbhost:1521:DATABASE
[...]
wls:/mydomain/config/JDBCConnectionPools/MyPool> get('TestConnectionsOn
  Release')
1
```

Listing 7

```
wls:/mydomain/config> datasource=create('MyDS', 'JDBCTxDataSource')
JDBCTxDataSource with name 'MyDS' has been created successfully.
wls:/mydomain/config> datasource.setJNDIName("MyDS")
wls:/mydomain/config> datasource.setPoolName("P6SPY Connection Pool")
wls:/mydomain/config> datasource.setEnableTwoPhaseCommit(true)
wls:/mydomain/config> datasource.addTarget(admin.myServer)
1
```

Listing 8

```
wls:/mydomain/config> cd ('JDBCTxDataSources')

wls:/mydomain/config/JDBCTxDataSources> ls()
drw-   MyDS

wls:/mydomain/config/JDBCTxDataSources> cd('MyDS')

wls:/mydomain/config/JDBCTxDataSources/MyDS> ls()
[...]
-rw-   PoolName                                   MyPool
[...]
wls:/mydomain/config/JDBCTxDataSources/MyDS> set('PoolName', MY_POOL_
  NAME)
```

FREE RESOURCE CD INCLU
information STORAGE+SECURITY journal
www.ISSJournal.com
In this issue:
IP Storage
SAN NAS
Disaster Recovery
Enterprise Security
Infosecurity
Microsoft RSA Security
SecureID for Windows
SAN NAS
Emerging technology trends and market maneuvers

>Linux >Java >Web Services >.NET >XML >Wireless >Storage >Security
The Premier Resource for Today's Corporate & IT Decision Makers
IT solutions >GUIDE
WWW.SYS-CON.COM
Top IT Solution Providers!
> Candle
> ClearNova
> Computer Associates
> Creative Science
> DralaSoft
> GraphOn
> InterSystems
> ITKO
> Merant
> O'Reilly
> Parasoft
> QuadBase
> SAP
> Wily Technology
> Zero G
TECHNOLOGIES YOU NEED NOW!
How to Manage Your Ideas Using Today's i-Technologies

Migrating from WebLogic 6.1 to 8.1

CATCHING THE FINE POINTS

By G.V.B Subrahmanyam

BEA's name is synonymous with the application server. Most applications running on Unix in North America run on BEA's WebLogic server, particularly financial apps. The current stable WebLogic 8.1 version does a lot more than earlier versions. Migrating to new versions is always challenging and risky, but is done for efficiency and new capabilities, and to stay current with a vendor's products and support.

Most applications developed on the WebLogic 6.1 used JDK 1.3 or earlier. A move from WebLogic 6.1 to 8.1 means making our applications compatible with JDK 1.4.1 or higher since WebLogic 8.1 shipped with JDK 1.4. Most of the common migration hazards are discussed at bea.com. I am going to share few personal observations.

These will be our discussion points.

- 6.1 to 8.1 migration, the current application scenario
- Common upgrade issues
- Getter and setter methods
- Serializable objects
- WL cache
- Tweak JVM memory usage
- Page directive with language attributes
- Java.net.SocketException: Too many open files
- Miscellaneous issues

6.1 to 8.1 Migration - The Current Application Scenario

Our application will be migrated to WebLogic 8.1. The current environment is:
Operating System: Solaris 5.8 P4
WebLogic Server 8.1P2
Java Release 1.4.2_06
Actuate 7.1SP6
Popular RDBMS and it has no impact on migration
Struts1.1 Framework
Model II Web Architecture

Common Upgrade Issues

Common issues in migrating from WebLogic 6.1 to 8.1 like class path, system path, environmental variables, data sources, deployment, JSP parsing, security, server configuration, and XML parsing are discussed in length at www.bea.com.

For example, take XML parsing. Unlike WebLogic 6.1 and earlier, WebLogic 8.1 doesn't tolerate encoding errors. So modify encoding from Unicode to utf-16.

Also there are no more JSP recompilations so the following lines have to be removed from the web.xml.

```
  <servlet>
    <servlet-name>JSPClassServlet</servlet-name>
    <servlet-class>WebLogic.servlet.JSPClassServlet</
servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>JSPClassServlet</servlet-name>
    <url-pattern>*.JSP</url-pattern>
  </servlet-mapping>
```

Getter and Setter Methods

Applications running under WebLogic 8.1 use JDK 1.4 or later. So when we migrate from WebLogic 6.1 to 8.1, we are obviously moving to JDK 1.4.1 or later. The later versions of JDK adhere to stricter Java specifications, which means that any improper code that may have worked well under earlier JDKs can cause errors in the JDK 1.4.1 and later.

For example, there was an error in using tag library uri='/WEB-INF/tlds/taglib.tld' prefix='j2ee': There is no setter method for property 'Items' for Tag class 'com.sun.j2ee.apps.application.taglib.list.SearchListTag' probably occurred because of an error in /template.JSP line 8: <%@ taglib uri="/WEB-INF/tlds/taglib.tld" prefix="j2ee" %>

This failure occurs in JSPs and beans that are subject to the Introspector API. One good example is getter and setter methods. If the setter is an integer, the getter must be an integer too, and can't be a string.

Serializable Objects

To support the replication of an Http session in memory, all object data have to be serializable. Otherwise session replication would fail. When debugging is enabled, WebLogic outputs a warning message indicating that the session hasn't replicated:

```
"Failed to replicate non-serializable object"
```

Author Bio:
G.V.B. Subrahmanyam is a consultant with Citigroup technologies. He holds an MTech and a PhD in technology from IIT Kharagpur, India. He also has a Masters in software systems from BITS Pilani, India.

Contact:
subrahmanyam.vb.gampa@citigroup.com or gvbsvv@gmail.com

A LIMITED TIME SAVINGS OFFER FROM *SYS-CON MEDIA*

SUBSCRIBE TODAY TO MULTIPLE MAGAZINES

AND SAVE UP TO $340 AND RECEIVE UP TO 3 FREE CDs!*

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

3-Pack
Pick any 3 of our magazines and save up to $210.00
Pay only $99 for a 1 year subscription plus a FREE CD
- 2 Year – $179.00
- Canada/Mexico – $189.00
- International – $199.00

6-Pack
Pick any 6 of our magazines and save up to $340.00
Pay only $199 for a 1 year subscription plus 2 FREE CDs
- 2 Year – $379.00
- Canada/Mexico – $399.00
- International – $449.00

9-Pack
Pick 9 of our magazines and save up to $270.00
Pay only $399 for a 1 year subscription plus 3 FREE CDs
- 2 Year – $699.00
- Canada/Mexico – $749.00
- International – $849.00

CALL TODAY! 888-303-5282

Pick a 3-Pack, a 6-Pack or a 9-Pack

☐ 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.

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.

☐ **LinuxWorld Magazine**

U.S. - Two Years (24) Cover: $143	You Pay: $79.99 /	Save: $63 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $39.99 /	Save: $32
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

☐ **JDJ**

U.S. - Two Years (24) Cover: $144	You Pay: $99.99 /	Save: $45 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $69.99 /	Save: $12
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $89.99 /	Save: $40
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

☐ **Information Storage + Security Journal**

U.S. - Two Years (24) Cover: $143	You Pay: $49.99 /	Save: $93 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $39.99 /	Save: $39
Can/Mex - Two Years (24) $168	You Pay: $79.99 /	Save: $88 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $49.99 /	Save: $34
Int'l - Two Years (24) $216	You Pay: $89.99 /	Save: $126 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $59.99 /	Save: $48

☐ **Wireless Business & Technology**

U.S. - Two Years (12) Cover: $120	You Pay: $49.00 /	Save: $71 + FREE $198 CD
U.S. - One Year (6) Cover: $60	You Pay: $29.99 /	Save: $30
Can/Mex - Two Years (12) $120	You Pay: $69.99 /	Save: $51 + FREE $198 CD
Can/Mex - One Year (6) $60	You Pay: $49.99 /	Save: $10
Int'l - Two Years (12) $120	You Pay: $99.99 /	Save: $20 + FREE $198 CD
Int'l - One Year (6) $72	You Pay: $69.99 /	Save: $2

☐ **MX Developer's Journal**

U.S. - Two Years (24) Cover: $143	You Pay: $49.99 /	Save: $93 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $39.99 /	Save: $32
Can/Mex - Two Years (24) $168	You Pay: $79.99 /	Save: $88 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $49.99 /	Save: $34
Int'l - Two Years (24) $216	You Pay: $89.99 /	Save: $126 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $59.99 /	Save: $48

☐ **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

☐ **WebSphere Journal**

U.S. - Two Years (24) Cover: $216	You Pay: $129.00 /	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: $189.00 /	Save: $75
Int'l - One Year (12) $132	You Pay: $129.99 /	Save: $2

☐ **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

☐ **WLDJ**

U.S. - Four Years (24) Cover: $240	You Pay: $99.99 /	Save: $140 + FREE $198 CD
U.S. - Two Year (12) Cover: $120	You Pay: $49.99 /	Save: $70
Can/Mex - Four Years (24) $240	You Pay: $99.99 /	Save: $140 + FREE $198 CD
Can/Mex - Two Year (12) $120	You Pay: $69.99 /	Save: $50
Int'l - Four Years (24) $240	You Pay: $120 /	Save: $120 + FREE $198 CD
Int'l - Two Year (12) $120	You Pay: $79.99 /	Save: $40

*WHILE SUPPILES LAST. OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Subscribe Online Today www.sys-con.com/2001/sub.cfm

SYS-CON MEDIA

If this happens, find the page throwing the error and make sure that all the data put in the session is serializable. Every object stored in the session has to implement a serializable interface for replication to happen.

WL Cache

The cache always has to be removed from each node for WebLogic to pick up any revised code using scripts. (Normally there are multiple nodes in a production environment for contingency, fault tolerance, and load balancing.) The WebLogic 8.1 server cache tag has attributes such as timeout, scope, key, async, name, size, and flush. When flush is set to true, the cache is flushed. A simple cache tag is shown below:

```
<wl:cache>
<!--the content between these tags will only be
 refreshed on server restart-->
</wl:cache>
```

Using a cache tag forces the cached values to be erased every time fresh values are calculated. The cache must be named with the name attribute. The size attribute will flush all values; the key attribute will flush specific values. WebLogic 8.1 has two versions of cache and this gives us the option of using the soft references for caching to avoid excessive use of the system memory. However, we can't use soft references when WebLogic is running in the Hotspot VM.

Tweak JVM Memory Usage

If the new environment has a JVM with a limited heap size then the files will have to be refactored. If we try to incorporate lots of content on a JSP page, it will have JVM memory problems. We all know that the JSP gets compiled into a servlet as a single method. If there are more tags, then the servlet method will be too big for the JVM to understand. In that case, the user will see an error saying, "Illegal target of jump or branch." It means that the JSP has exceeded the size and has to refactor them.

One solution is to increase the JVM's heap size to a higher value if more memory is required to run the files. Increasing the heap size can reduce delay and add stability. So, on the JVM options line in the binTogether.bat file, change -Xms64m to –Xms256m (or higher).

Another solution is to break a big page into many smaller pages and bring them together at runtime using include. Using <%@include> won't work because that's a compile-time include.

Still another approach is to look for places to save the tags. For example, the tag <html:option>, which was extended to let the text display. You can replace this:

```
<Html:option ... ><bean:message key="foo"/></
html:option>
```

with this:

```
<html:option ... key="foo"/>
```

The more cases of this type there are the better the situation. Besides the <html:option> tag, some other struts tags are shortcuts to including the text. You might also consider replacing <html:option> sequences with <html:options> if you can build an appropriate collection upfront.

One resolution is to have our own tags. Changing some frequently used tags by extending BodyTagSupport to extending Tag Support and marking them with <body content>empty</body content> in the .tld file significantly reduces generated code size. This may not always be the way out, but it helps.

Whenever the JSP is too big, one must set the variable com.sun.tools.javac.main.large-branch to true.

In that case, you have to use JRocket, which is bundled with WebLogic 8.1. It can handle memory better than the JDK.

Page Directive with Language Attribute

In some JSP code, the value for attribute language is specified multiple times. This worked in WebLogic 6.1 but in migrating code to WebLogic 8.1, we can specify the values only once or else it will throw an JSPException as required in the J2EE specification.

Java.net.SocketException: Too Many Open Files

Just after converting to WebLogic 8.1, the server is going to go down and send the following message. It may not be related to the migration, but it has to be resolved.

```
<Critical> <WebLogicServer> <BEA-000204>
<Failed to listen on port 7001, failure count:
1, failing for 0 seconds, java.net.Socke tExcep-
tion: Too many open files>
```

An immediate step to take is to increase the rlim_fd_max to 8192 in /etc/system. But this can lead to the usage going up to 7000 and may not prove a solution.

The best thing to do is fine-tune the TCP parameters on the operating system.

Java.lang.OutOfMemoryError

The error is Runtime Exception trying to execute a procedure using Sybase Pool in WLS 8.1 SP3. This error occurrs when we use WebLogic 8.1 SP3. BEA recognized this issue and it is now fixed in the WebLogic 8.1SP4 (CR196372). For those who would like to use 8.1SP3, BEA is providing a patch by means of a JAR file.

Miscellaneous

a) *Page directive with language attribute*
In some JSP code in some files the value for the attribute language is specified multiple times. This worked in WebLogic 6.1 but in migrating to WebLogic 8.1, we have to be sure to specify the values only once or else it will throw an JSPException in compliance with the J2EE specification.

b) *Deploying the application in Staging Mode*
WebLogic 8.1 lets applications be deployed in multiple modes depending on the situation. In deploying on a single server the domain always use the no_stage mode pattern.

c) *Defining context root*
Ours is a standalone application, the snippet below should be added to the web.xml.

```
<context-root>isoft/iSoft</context-root>
```

Summary

In all applications, code compliance with J2EE specification is mandatory regardless of the application server guidelines.

References

- www.serverwatch.com/tutorials/article.php/10825_3418341_2
- www.bea.com/framework.JSP?CNT=wp_abst00014.htm&FP=/content/news_events/white_papers
- http://e-docs.bea.com/wls/docs81/upgrade/quickupgrade.html
- http://developer.bea.com/products/wlserver81/articles/JSP_reloaded.JSP
- http://support.bea.com/support_news/product_troubleshooting/Too_Many_Open_Files_Pattern.html
- http://e-docs.bea.com/wls/docs81/notes/resolved_sp04.html
- http://e-docs.bea.com/wls/docs81/deployment/overview.html
- http://e-docs.bea.com/wls/docs81/webapp/WebLogic_xml.html

– continued from page 11

A Migration Strategy for WebLogic Server 5.1 to WebLogic Server 8.1

8. Import the Certificate-Key pair xenon-cert.pem/ xenon-key.pem files into a KeyStore file using the following steps:
   a. Copy the xenon-cert.pem into xenon-cert-chain.pem
   b. Append the contents of the VerisignRSACA root certificate (or any other root certificate) in Base 64
   format to xenon-cert-chain.pem
   c. Execute the following command to create a new Private KeyStore:

```
java utils.ImportPrivateKey xenon.ks file-
pass xenonAlias keypass xenon-cert-chain.
Pem xenon-key.pem
```

Check if the KeyStore is valid by opening it in the KeyTool GUI. The SSL setup in WebLogic Server 8.1 can be done using the administration console.

Summary

WebLogic Server 8.1 SP1 provides a stable environment for Web applications and enterprise archive applications. The WebLogic Server Administration Console, WebLogic Workshop, and the numerous other tools that are part of WebLogic Server 8.1 provide additional value when migrating or developing new applications on WebLogic Server 8.1. The steps mentioned in this article are some of the crucial tasks that architects and developers should be aware of while migrating or developing new applications on WebLogic Server 8.1.

Acknowledgements

I would like to thank Venkataraman Sridharan, Pankaj Khandar, and Asha Veerabhadraiah for their assistance in preparing this article. Special thanks also to Linnae DeSanto and Venkataraman Sridharan for providing feedback and peer, editorial, and technical reviews.

Integrating an ASP.NET Application with J2EE PART 1

PLATFORM INTEROPERABILITY

By Blair Taylor

We all know that the .NET platform offers a great set of tools for building robust Web applications. There are times, however, when as .NET developers we need to understand how we can integrate the great features of .NET with other platforms and technologies. We often find our clients using both .NET and J2EE technologies successfully in their architectures.

I faced a situation such as this recently and I would like to share some of my experiences with you in this two-part article. Part 1 will discuss interoperability between ASP.NET and BEA WebLogic 8.1, and how to use XML Schemas to transfer data between the platforms. Part 2 will discuss how to properly process SOAP Exceptions, the uploading and viewing of binary data, and how we handled page navigation and application workflow. The source code for this article can be found at http://sys-con.com/weblogic/sourcec.cfm.

On a recent project, the client decided that they wanted to use J2EE technology (specifically BEA WebLogic 8.1) to support their core legacy applications. The new Web application would let vendors enter information that would be validated and transferred directly into the core application database. WebLogic would provide the business functionality via Web Services. The client had several applications developed in C# using Visual Studio.NET and wanted to take advantage of the productive features in ASP.NET to develop the presentation layer of the application.

There are various technologies we could use so .NET could communicate with J2EE. The most popular one is XML Web Services. The main benefits of Web Services are the industry support, tool support, and ease of development.

There are also several products such as Ja.NET (http://ja.net.intrinsyc.com/ja.net/info) or Janeva (www.borland.com/janeva) that bridge .NET and J2EE environments to allow binary communications and share native data types and stateful communication. These products bridge .NET and J2EE by configuring .NET Remoting to behave like a Java RMI/Corba client or server. Bridging .NET and J2EE provides more robust functionality and performs better than XML Web Services. The big drawbacks in using a bridging technology are the complexities of learning and configuring the bridging product and the cost of licensing it. We decided to use XML Web Services because the web application didn't need stateful communication and speed wasn't as high a priority as development time and cost.

The solution was to use ASP.NET Web Forms in the presentation layer communicating with Web Services hosted on WebLogic, which would provide the business services and data persistence. The presentation tier would display, capture, and perform simple domain validations on the data. The business tier would apply the business rules and the data tier would persist data to a SQL Server database. The ASP.NET Web Forms would communicate with the WebLogic business layer using WebLogic Web Services.

WebLogic Web Services are built using the WebLogic Web Services toolkit. It lets you expose a Java object as a Web Service without additional coding. WebLogic Web Services support the following components as the source of a Web Service:

- A method of a stateless session Enterprise Java Bean (EJB)
- A method of a Java class
- A JMS message consumer or producer

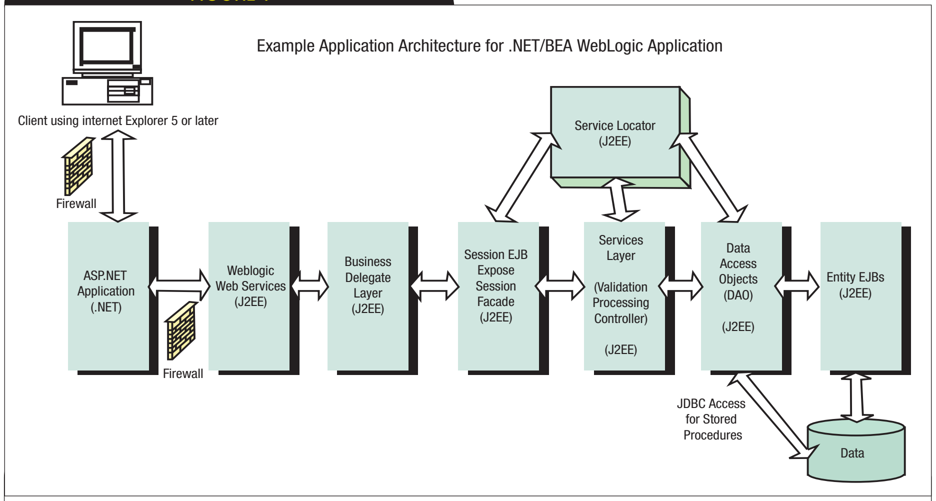
BEA recommends that you implement your Web Service operation with only a stateless-session EJB or a Java class, and not with a JMS consumer or producer.

In our application, we used stateless session beans as a facade to the business logic and data persistence logic. The pattern we followed is the Session Facade pattern, which provides a single interface for application functionality and allows for course-grained access to business objects. You can find more information on the facade pattern at http://java.sun.com/blueprints/patterns/SessionFacade.html.

Author Bio:
Blair Taylor is president of JustWebSolutions.com, a Canadian company specializing in the architecture and development of distributed systems. Blair has authored several publications covering client-server and distributed technologies and is certified in both Java and .NET technologies. Blair can be reached via e-mail at feedback@justwebsolutions.com or at www.justwebsolutions.com.

Contact:
feedback@justwebsolutions.com

FIGURE 1


Example Application Architecture for .NET/BEA WebLogic Application
Client using internet Explorer 5 or later
Firewall
ASP.NET Application (.NET)
Firewall
Weblogic Web Services (J2EE)
Business Delegate Layer (J2EE)
Session EJB Expose Session Facade (J2EE)
Service Locator (J2EE)
Services Layer (Validation Processing Controller) (J2EE)
Data Access Objects (DAO) (J2EE)
Entity EJBs (J2EE)
JDBC Access for Stored Procedures
Data

Application architecture

We also implemented an intermediate class to act as a business delegate that decouples business components from the code that uses them. Specifically, the business delegate class assisted in implementing exception handling by letting us throw application exceptions from Session EJBs and catch them in the business delegate class that then builds a custom SOAP exception. This is discussed in more detail in the section on SOAP exception handling and you can find more information on the business delegate pattern at http://java.sun.com/blueprints/patterns/BusinessDelegate.html.

Figure 1 shows the overall architecture of our application.

There were several challenges we faced. In this article I'll discuss some of these issues, how we decided to tackle them in our implementation, and our advice to anyone using a similar architecture.

We identified the following issues:

- We needed to ensure that the Web Services we developed were interoperable between ASP.NET and BEA WebLogic 8.1.
- We needed to determine the best method to transfer data via Web Services between the platforms.
- We needed to determine how we were going to do exception handling between the two platforms.
- We needed to upload and view images that would require the transfer and retrieval of binary information (images) via Web Services.
- We wanted to devise a plan for handling page navigation and application workflow.

Interoperability between .NET & BEA WebLogic 8.1

The biggest appeal in using XML Web Services is the true interoperability offered between different platforms. To discuss true interoperability, we need to discuss some of the technologies behind Web Services.

The SOAP specification dictates that the contents of a message sent via a Web Service must be XML-formatted, but the specific format used is open to the vendor. The two predominant styles WebLogic supports are RPC-style and document-style. An RPC-style message is one in which the SOAP messages contain parameters and return values. A document-style message is one in which the SOAP messages contain XML documents.

The default style for .NET XML Web Services is document. The default style for the WebLogic Web Services toolkit is RPC (as it is for the Microsoft SOAP Toolkit v2). The preferred style is a source of debate. Many feel that document style is more flexible. Both styles are transparent to an ASP.NET developer since the proxy class generated by wsdl.exe supports both. Since interoperability was the key issue, we chose the RPC style, which has a longer history of support in most toolkits, and also makes the Web Service appear like a local API to the caller. RPC style also let us design Web Services interfaces similar to the standard interfaces we would use in Java or .NET.

There are two styles of encoding native data types in the XML sent as part of a SOAP message – SOAP encoding and literal encoding. SOAP encoding uses the SOAP specification to map native data types to XML. The SOAP specification defines native

TABLE 1

XML Schema Data Type	Equivalent Java Data Type
boolean	boolean
byte	byte
Short	short
int	int
long	long
float	float
double	double
integer	java.math.BigInteger
decimal	java.math.BigDecimal
string	java.lang.String
dateTime	java.util.Calendar
base64Binary	byte[]
hexBinary	byte[]
duration	weblogic.xml.schema.binding.util.Duration
time	java.util.Calendar
date	java.util.Calendar
gYearMonth	java.util.Calendar
gYear	java.util.Calendar
gMonthDay	java.util.Calendar
gDay	java.util.Calendar
gMonth	java.util.Calendar
anyURI	java.lang.String
NOTATION	java.lang.String
token	java.lang.String
normalizedString	java.lang.String
language	java.lang.String
Name	java.lang.String
NMTOKEN	java.lang.String
NCName	java.lang.String
NMTOKENS	java.lang.String[]
ID	java.lang.String
IDREF	java.lang.String
ENTITY	java.lang.String
IDREFS	java.lang.String[]
ENTITIES	java.lang.String[]
nonPositiveInteger	java.math.BigInteger
nonNegativeInteger	java.math.BigInteger
negativeInteger	java.math.BigInteger
unsignedLong	java.math.BigInteger
positiveInteger	java.math.BigInteger
unsignedInt	long
unsignedShort	int
unsignedByte	short
Qname	javax.xml.namespace.QName

WebLogic built-in data types

data types that are supported and you must use one of these data types. Literal encoding uses XML Schema to include a definition for all non-standard data types in the XML message itself. Literal encoding is more flexible in this instance.

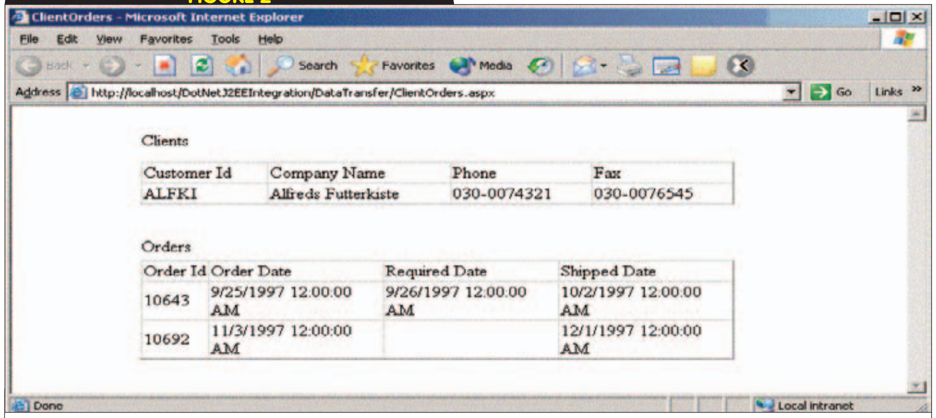
RPC-style WebLogic Web Service operations must use SOAP encoding. Document-style WebLogic Web Service operations must use literal encoding. All operations in a single WebLogic Web Service (class) must be either RPC-style or document-style; WebLogic Server doesn't support mixing the two styles in the same implementation. Since we were using an RPC style of Web Service, all the Web Services in WebLogic would use SOAP encoding.

In the WebLogic application, we used built-in data types exclusively to minimize the development effort. Built-in data types are those specified by the JAX-RPC specification. When using built-in data types, WebLogic automatically handles the conversion of the data between its XML and Java representation. If the Web Service uses a non-built-in data type as a parameter or return value, you must create a serialization class to convert the data between Java and XML.

Table 1 lists the built-in data types WebLogic supports.

For the most part, we encountered few incompatibility issues consuming WebLogic 8.1 Web Services in an ASP.NET application. One issue of note is the use of SOAP encoding. While the WS-I Basic Profile 1.0 (BP) requires the RPC style and literal encoding, using RPC style in WebLogic requires SOAP encoding which violates the WS-I Basic Profile 1.0. For more information on the WS-I Basic Profile 1.0, see the sidebar, "WS-I Basic Profile 1.0." We should also note that using WebLogic Web Services, you can't expose static methods as a Web Service; they must be class-level methods.

FIGURE 2


ClientOrders - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/DotNetJ2EEIntegration/DataTransfer/ClientOrders.aspx
Go Links
Done
Local intranet

Clients

Customer Id	Company Name	Phone	Fax
ALFKI	Alfreds Futterkiste	030-0074321	030-0076545

Orders

Order Id	Order Date	Required Date	Shipped Date
10643	9/25/1997 12:00:00 AM	9/26/1997 12:00:00 AM	10/2/1997 12:00:00 AM
10692	11/3/1997 12:00:00 AM		12/1/1997 12:00:00 AM

ClientOrders.aspx ou

WS-I Basic Profile 1.0

A key part of true interoperability is the WS-I Basic Profile 1.0. The Web Services Interoperability Organization (WS-I) is an organization of Web Service vendors that includes Microsoft Corp., BEA Systems, Sun Microsystems, and many other industry-leading vendors. The goal of the WS-I is to promote Web services interoperability. In keeping with this goal, the WS-I has produced the WS-I Basic Profile 1.0 (BP). The BP defines rules to assist in areas of the Web services specifications (XML 1.0, SOAP 1.1, WSDL 1.1, and UDDI 2.0) in which the specification is unclear or ambiguous. The BP defines constraints and clarifications to the base specifications to enhance interoperability of Web services between different vendors.

Examples of some of the rules imposed by the BP include:

- precluding the use of SOAP encoding
- requiring the use of HTTP binding for SOAP
- requiring the use of HTTP 500 status response for SOAP Fault messages
- requiring the use of HTTP POST method
- requiring the use of WSDL1.1 to describe the interface of a Web service
- requiring the use of RPC/Literal or Document/Literal forms of the WSDL SOAP binding
- precluding the use of solicit-response and notification style operations
- requiring the use of WSDL SOAP binding extension with HTTP as the required transport

So what does this mean to a Web services developer? Well, many of the rules deal with restrictions that are relevant to the tool developers rather than to the developer. For example, most tools implicitly require the use of HTTP with Web services by default and a developer would need to take extra steps (and understand the implications) to use a transport other than HTTP to build Web services.

Transferring Data via Web Services

Since the application needed to call non-.NET Web Services, the mechanism that we used to transfer data couldn't use data types or objects specific to .NET. In previous projects we had passed DataSets through Web Services and bound them to ASP.NET components in our Web applications. Since WebLogic has no concept of a DataSet, we couldn't use this approach. We considered a number of alternatives.

We considered building a mechanism to generate XML that would be identical to the XML generated from a DataSet. This would require generating an XML schema dynamically based on the data being passed and then the generation of XML data based on the dynamic schema. We decided against this solution because we felt it would take a great deal of effort to code and was susceptible to future changes in ADO.NET

Another option was to use value objects to pass the data. Value objects are based on the "Value Object" Pattern and represent simple classes with properties and accessor/mutator (get/set) methods. Value objects are often used in J2EE architectures to pass data between tiers. In this case, a value object or collection of value objects would be instantiated in a WebLogic Web Service and data would be stored in these classes. The data would be received in the ASP.NET application as a proxy object, which is a client representation of the Java classes. The classes must use primitive data types (or other collection objects such as ArrayLists that are recognized by both environments).

We decided against this solution because of the effort involved in parsing the data passed via the proxy objects. Under this approach a .NET developer would need to write code to parse the values from the proxy object and put them in a DataSet. This is relatively simple to code but it's a rather tedious and unnecessary step. The code would also need to change each time the data returned from the Web Service changed.

We wanted to find a solution that would involve minimal coding, use technologies available in both .NET and WebLogic, and would let ASP.NET developers bind objects such as DataGrids or DataLists directly to the data returned from the Web Service (as can be done when passing DataSets from a .NET Web Service).

The solution we decided on was to use XML schemas as the root of our solution. For each result set, we built an XML schema to define the data in the result set. To generate the XML data in the WebLogic environment, we used a product called XMLBeans. XMLBeans is a Java-XML binding tool that lets you to generate Java objects based on an XML schema easily. It generates an object representation of an XML schema in Java classes. BEA uses XMLBeans extensively in WebLogic itself. XMLBeans is an open source product originally developed by BEA and released to the open source community via Apache. You can learn more about XMLBeans at http://dev2dev.bea.com/technologies/xmlbeans/index.jsp and you can download the BEA version at http://workshop.bea.com/xmlbeans/XsdUpload.jsp.

Using XMLBeans, we generated a set of Java classes to contain our data and then used the XMLText method to get an XML document instance based on the object data. We could then pass this XML string via a Web Service to our ASP.NET application.

In the ASP.NET application, a developer could instantiate a DataSet, load in an XML schema to define the structure of the DataSet, and then load the XML document into the DataSet via the LoadXML method.

In our example, which is based on the Northwind database, we build a schema to represent a set of client and order data. The ClientOrders.xsd schema is shown in Listing 1. To generate a set of Java objects based on the ClientOrders.xsd schema, you need to download and install the XMLBeans toolkit. As with all Java tools, you need to set the correct location of the Java SDK before running it. If you've installed a Java product (such a JBuilder or WebLogic), an SDK will likely be installed. Listing 2 shows a sample batch file that you can use – you will need to set the "JAVA_HOME" variable to point to your Java SDK location. The sample batch file expects to find the XMLBeans library (xbean.jar) in the default installation location (C:\xmlbeans).

Once you've run the XMLBeans tool, you'll have a .jar file that contains the classes that are the object representation of a ClientOrders.xsd XML schema.

The output should look like this:

```
C:\xmlbeans\output>scomp ClientOrders.xsd
Loading schema file ClientOrders.xsd
Time to build schema type system: 3.655 seconds
Time to generate code: 12.107 seconds
Compiled types to xmltypes.jar
```

Rename the xmltypes.jar file to ClientOrders.jar. The Java code sample simply instantiates the Java classes and uses the mutator methods to set sample values based on the Northwind database. In a real-life situation, you'd likely use a Java technology (such as JDBC or EJB) to access the database and get live data. After building the appropriate Java objects, the code calls the XMLText method to get an XML document instance representing the client and order data. The XML document instance can be validated using the ClientOrders.xsd XML schema. We then pass this XML string as the result of the Web Service generateTestData method.

Listing 3 shows the sample Java code to use the ClientOrders.jar to build an XML document instance.

The code for the ASP.NET client page first instantiates a Web Service proxy and makes the call to the WebLogic Web Service.

```
// Call J2EE web service and obtain xml document
weblogic.BusinessDelegate proxy = new weblogic.
BusinessDelegate();

// Obtain XML document instance
string xmlDoc = proxy.generateTestData();
```

The ASP.NET client page then defines a DataSet, reads the XML into a string, uses the ClientOrders.xsd schema to define the DataSet structure, and loads the DataSet using the XML string passed.

```
DataSet dsClientOrder = new DataSet();
dsClientOrder.EnforceConstraints = false;
DataTable dataTable = new DataTable();

string schemaRelativePath = "~/Schemas/Clien-
tOrders.xsd";
string schemaFullPath = HttpContext.Current.
Server.MapPath(schemaRelativePath);

StringReader stringReader = new
StringReader(xmlDoc);
XmlReader xmlReader = new XmlTextReader(string
Reader);
dsClientOrder.ReadXmlSchema(schemaFullPath);

dsClientOrder.ReadXml(xmlReader, XmlReadMode.
IgnoreSchema);
```

The ASP.NET client page then binds the DataSet to the dgClients and dgOrders datagrids.

```
// now bind to DataGrids
dgClients.DataSource = dsClientOrder;
dgClients.DataMember = "ClientOrder";
```

Article Information Box

List of Prerequisites

Some knowledge of Java and J2EE or BEA Weblogic
Some experience with ASP.NET and XML Web services
Some knowledge of XML and XML schemas

Level of Difficulty

2 (of 3)

Summary

This article discusses some of the issues that arise when using .NET with a J2EE application server such as BEA Weblogic. The tools used are Visual Studio.NET 2003 and BEA Weblogic Server 8.1. The article discusses several issues: integration between the two technologies; how to transfer data from Weblogic to ASP.NET so that data can be easily bound to ASP.NET components; proper exception handling within Web services; and how to store and retrieve binary data. The article also looks at the Apache Struts framework and how navigation within an ASP.NET application can be configured via an XML file in a manner similar to the Struts framework.

Background Information

- BEA Weblogic Web services: http://edocs.bea.com/wls/docs81/webserv/index.html
- XMLBeans: http://dev2dev.bea.com/technologies/xmlbeans/index.jsp
- XML technologies: www.w3.org/XML/, www.w3.org/XML/Schema
- SOAP tutorial: www.w3schools.com/soap/default.asp
- The Apache Struts Web Application Framework: http://jakarta.apache.org/struts/

Related Articles

- MSDN Web Services Development Center: http://msdn.microsoft.com/webservices/
- XML, SOAP and Binary Data whitepaper on MSDN: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/infoset_whitepaper.asp
- The XML Files - A Quick Guide to XML Schema: http://msdn.microsoft.com/msdnmag/issues/02/04/xml/default.aspx
- Enterprise Interoperability: .NET and J2EE: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dotnetinteroperability.asp
- Using Soap Faults: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service09172002.asp

```
dgClients.DataBind();

dgOrders.DataSource = dsClientOrder;
dgOrders.DataMember = "Order";
dgOrders.DataBind();
```

Figure 2 shows the page after the code has completed.

We got the benefit of XML schema validation on our data, used an object representation of the data in WebLogic, and passed the data to the ASP.NET application in a way that made loading the data into a DataSet straightforward. We could then bind GUI objects such as a DataGrid or DataList directly to the DataSet table. This eliminates the need to parse the data passed by the Web Service. It also provides a flexible architecture in which the changes to the data passed to the ASP.NET application don't require changes to the interface. The data passed isn't validated by an interface, but it is validated via the XML schema.

Listing 1: ClientOrders.xsd

```
<?xml version="1.0" encoding="UTF-8"?><xs:schema
  targetNamespace="http://www.justwebsolutions.com/articles" xmlns:
  xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.
  justwebsolutions.com/articles" elementFormDefault="qualified"
  > <!-- define global elements for Client --> <xs:element
  name="CustomerId" type="xs:string"/>  <xs:element name="CompanyName"
  type="xs:string"/>  <xs:element name="Phone" type="xs:string"/>
<xs:element name="Fax" type="xs:string"/> <!-- define global
 elements for Order --> <xs:element name="OrderId" type="xs:
 int"/> <xs:element name="OrderDate" type="xs:date"/> <xs:
 element name="RequiredDate" type="xs:date"/>
 <xs:element name="ShippedDate" type="xs:date"/>  <!-- define Order
 element -->  <xs:element name="Order">   <xs:complexType>      <xs:
sequence>   <xs:element ref="OrderId"/>
 <xs:element ref="OrderDate"/>
 <xs:element ref="RequiredDate"/>        <xs:element
ref="ShippedDate"/>
 </xs:sequence>   </xs:complexType> </xs:element>
 <!-- define Client element -->  <xs:element name="ClientOrder">   <xs:
complexType>      <xs:sequence>       <xs:element ref="CustomerId"/>
      <xs:element ref="CompanyName"/>       <xs:element ref="Phone"/>
      <xs:element ref="Fax"/>       <xs:sequence>          <xs:ele-
ment ref="Order"/>        </xs:sequence>      </xs:sequence>    </xs:
complexType>  </xs:element> <!-- define schema elements -->  <xs:element
name="ClientOrders">    <xs:complexType>      <xs:sequence>       <xs:
element ref="ClientOrder"/>      </xs:sequence>    </xs:complexType>
</xs:element></xs:schema>
```

Listing 2: XMLBean generation script

```
@rem Schema compiler
@rem
@rem Builds XBean types from xsd files.

@echo off

rem *** SET JAVA_HOME to the location of the Java 1.4 SDK
set JAVA_HOME=C:\JBuilderX\jdk1.4
set PATH=%JAVA_HOME%\bin
set cp=C:\xmlbeans\xkit\lib\xbean.jar

java -classpath %cp% com.bea.xbean.tool.SchemaCompiler %*

:done
```

Listing 3: ClientOrdersDAO.java

```
package com.justwebsolutions.articles.dotnetj2ee.datatransfer;

import com.justwebsolutions.articles.*;
import java.util.Calendar;

public class ClientOrdersDAO {

  public ClientOrdersDAO() {
  }

  public static String generateTestData() {

    // generate some data for client-orders
    ClientOrdersDocument xmlDoc = ClientOrdersDocument.Factory.
      newInstance();
    ClientOrdersDocument.ClientOrders clientOrder =
      xmlDoc.addNewClientOrders();
    ClientOrderDocument.ClientOrder client =
     clientOrder.addNewClientOrder();

    // add client information
    client.setCustomerId("ALFKI");
    client.setCompanyName("Alfreds Futterkiste");
    client.setFax("030-0076545");
    client.setPhone("030-0074321");
    // add order 1
    OrderDocument.Order order = client.addNewOrder();
    order.setOrderId(10643);
    Calendar cal = Calendar.getInstance();
    cal.set(1997, 8, 25);
    order.setOrderDate(cal);
    cal.set(1997, 8, 26);
    order.setRequiredDate(cal);
    cal.set(1997, 9, 2);
    order.setShippedDate(cal);

    // add order 2
    OrderDocument.Order order2 = client.addNewOrder();
    order2.setOrderId(10692);
    cal.set(1997, 10, 3);
    order2.setOrderDate(cal);
    cal.set(1997, 10, 31);
    order2.setShippedDate(cal);

    // obtain the XML document instance as a string and return
    return xmlDoc.xmlText();

  }

}
```

Need to solve performance problems in BEA WebLogic™ 8.1 applications?

Manage Complex Enterprise Applications
Acsera has the advanced technology needed to manage complex BEA WebLogic™ 8.1 applications in production.

100% Automatic Discovery and Blueprinting
Acsera discovers your WebLogic clusters, nodes, servers and deployed applications automatically in minutes.

Performance Management and Monitoring
Acsera is Performance Centric – it helps you deliver the high throughput and reliability your enterprise needs from WLI.

Complete Visibility into Business Processes
No more 'black box' applications in production. Acsera shows business processes and their complete hierarchy.

Root Cause Analysis on Running Applications
No more 'mystery performance problems' on WLI. Acsera helps you diagnose and isolate problems while in production.

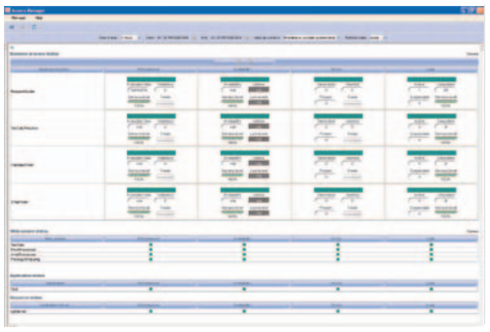
Manages Across Clusters
Your WLI applications deploy across nodes and clusters – don't be trapped into managing with server-by-server tools.

Results in Minutes
You can't afford to wait hours or days to find WLI problems and correct them – Acsera delivers results in minutes.

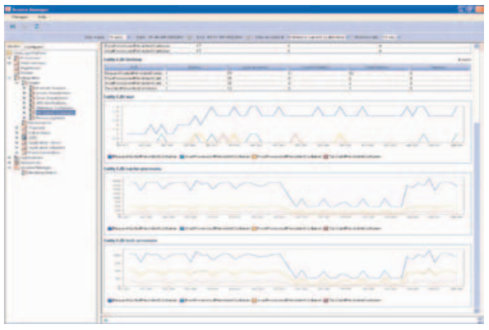
Attend a WebLogic Integration performance webinar: www.acsera.com/register

ACSeRA

Application Monitoring and Performance Management for BEA WebLogic™ 8.1


Performance Management Console
Shows application performance and health.

Application Blueprint Console
Gives 100% visibility into application hierarchy.


Root Cause Analysis Console
Isolates performance problems in minutes.

Motive brings an enlightened approach to application management, building management intelligence into the application itself. Only Motive transcends the fragmented, disjointed reality of traditional application management, to deliver the visibility, insight and automation that support and development teams need to keep applications running smoothly.

This is no mere vision for the future. It's here now in application management products from Motive. Learn more about Motive's breakthrough approach in a new white paper about built-in management at www.motive.com/within1.

motive®

www.motive.com