



ARE ALL JDBC DRIVERS CREATED EQUAL?

MAY/JUNE 2004

WLDJ™

WWW.WLDJ.COM

**THE LEADING
INDEPENDENT
MAGAZINE
FOR WEBLOGIC™
PROFESSIONALS**

The BEA WebLogic Message Bridge

**TRANSFER
MESSAGES
BETWEEN JMS
PROVIDERS ...32**

PLUS...

Workshop on My Mind...4

Instrumenting a
Java Page Flow Using
JMX Technology...6

An Architectural
Blueprint:
The Reasons for
Building a Model...16

Considering MySQL?
Read On...20

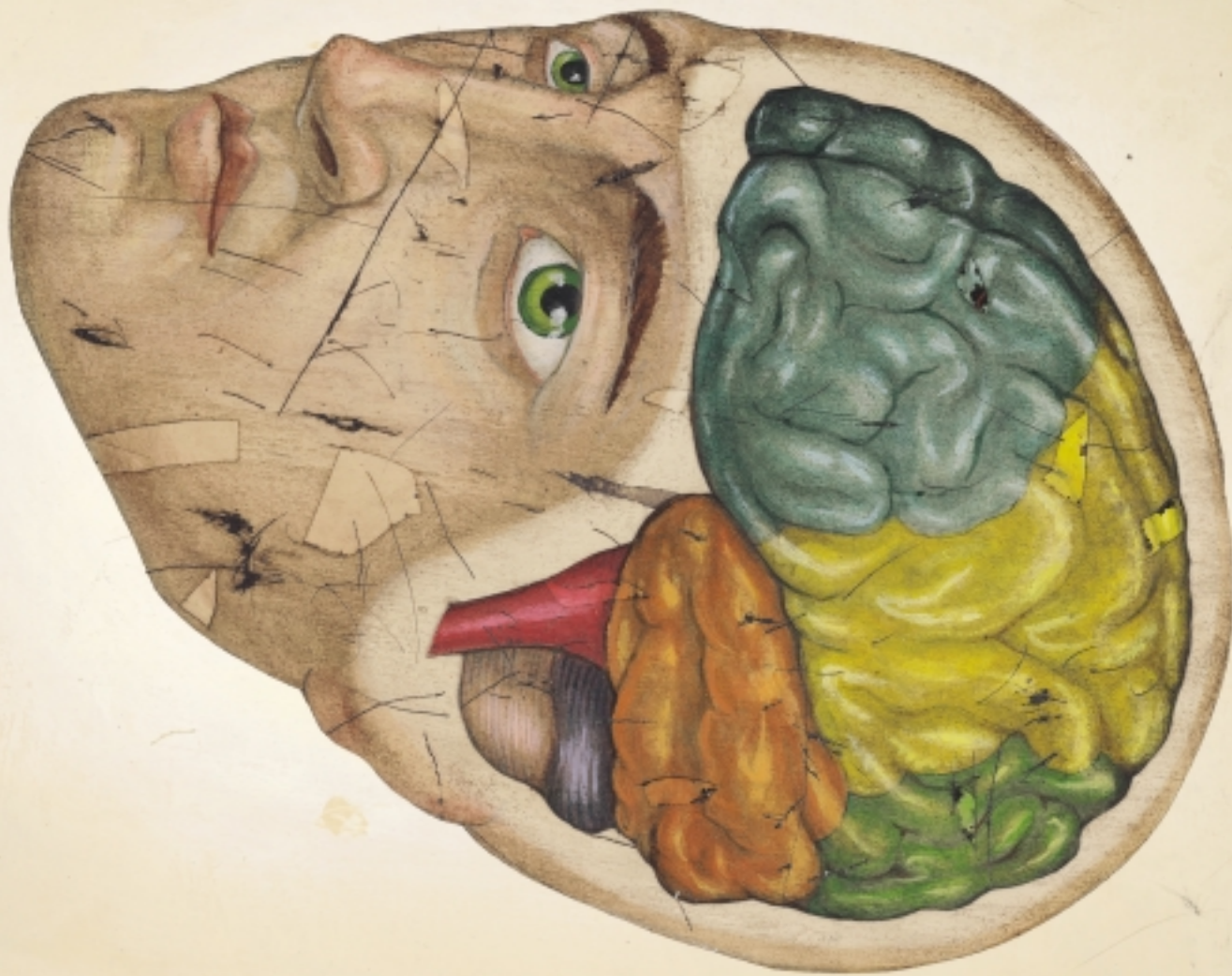


SUPPORT:
*JMX
Debugging*
page 28

MANAGEMENT:
*Application Management
with WebLogic Server
for Developers* page 40

TRANSACTIONS:
*Driving You to
Distraction?*
page 43

ADMINISTRATION:
*The Promise of
Utility Computing
Today* page 45



DO YOU HAVE A BRAIN?

CEREBRUM: CONTROLS THOUGHT.

*What is 2+2? What color is the sky?
If you can answer these, then you have a cerebrum.*

MEDULLA OBLONGATA: CONTROLS INVOLUNTARY FUNCTIONS.

*Check your pulse. Do you have one?
Then you have a medulla oblongata, too.*

BROCA'S AREA: CONTROLS LANGUAGE.

*Say the following sentence: "Frankly, Hector, I'm a bit surprised."
Did it work? Then you have a Broca's Area.*

PITUITARY GLAND: CONTROLS HORMONES.

*When you were about 13, did your voice change?
Did you grow hair in special places?
Then you've got a pituitary gland, friend.*

YOU HAVE A BRAIN.

So why spend so much time on mindless coding?

Download Weblogic Workshop Now.

BEA WebLogic Workshop™ 8.1 is a more efficient way to develop in J2EE.
And that means less grunt work and more real work. dev2dev.bea.com/medulla.





FROM THE EDITOR



Workshop on My Mind

BY JOE MITCHKO


Over the past several months, I've had the opportunity to interface with several BEA WebLogic project teams and ask how they do their development. One question I usually bring up, mainly out of curiosity, is whether or not they decided to use BEA WebLogic Workshop as part of their overall development strategy. As you may already know, Workshop is designed to streamline the overall development effort and can provide substantial improvements in programmer productivity along with streamlining the configuration and deployment process. In theory, the decision to use Workshop as the IDE of choice for BEA WebLogic development should be a no-brainer. But what is really happening out in the field?

Well, the result of my impromptu, and albeit unscientific, survey shows something quite unexpected. In one case, the project team was involved in developing a portal-like application that would be deployed to a WebLogic 8.1 application server. Okay, you would think using BEA WebLogic Portal along with the Java Page Flow designer tool, both integrated quite nicely in Workshop, would be the logical choice. To my surprise, the architects decided to develop the application using Struts and were not planning to use WebLogic Portal at all. Their reasons were as follows. They wanted a pure J2EE design with the ability to deploy the application to any J2EE-compliant Java application server, including Tomcat. I guess they weren't aware that you aren't locked into using the BEA WebLogic Server for Workshop-developed front-end applications.

In another encounter with a WebLogic development team, I found a similar shying away from using Workshop, but for other reasons. In this case, the development team was involved more on the back end with message processing using Web services and EJB components (includ-

ing message-driven beans). The reasons this time had to do with the performance and overall stability of the Workshop IDE itself. According to one of the developers, it has nice time-saving features, but it tends to be a resource hog, and bogs the system down. It also tends to freeze up from time to time. Consequentially, the development team is using JBuilder instead.

So, what's going on here? Did I just happen to stumble on a few development teams who aren't completely sold on the full capabilities of the BEA WebLogic product line, or is it more of a trend? Pondering the situation, I thought of a few reasons that would explain it. Since J2EE and associated technologies like Web services are based on open standards, each development team is ultimately free to pick and choose what they feel would be the best combination of tools and design techniques to get the job done. I would think in a number of cases development teams will choose solutions they are familiar with and that have a proven track record. An architect may also decide to go with an open source solution, like the team using Struts, because of the unique and robust design brought on by the contributions from the best and the brightest in the industry. Now, if this were Microsoft, there wouldn't be too many options when it comes to development. You basically use Visual Studio along with any plugins you can find, and that's it. In the J2EE world, you can get by with a simple text editor and a set of JAR files downloaded from Apache. Opposite ends of the spectrum.

I hope in the future more development teams will catch on and discover the robust capabilities of BEA WebLogic Workshop and that BEA will improve Workshop efficiency in upcoming releases so that developers with less than stellar workstations can make use of it. 

wldj THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC PROFESSIONALS

EDITORIAL ADVISORY BOARD
LEWIS CIRNE, KEVIN JONES,
TYLER JEWELL, WAYNE LESLEY LUND, CHRIS PELTZ,
SEAN RHODY, CARL SJOGREEN

FOUNDING EDITOR

PETER ZADROZNY

EDITOR-IN-CHIEF

JOE MITCHKO

PRODUCT REVIEW EDITOR

JASON SNYDER

EXECUTIVE EDITOR

GAIL SCHULTZ

EDITOR

NANCY VALENTINE

ASSOCIATE EDITORS

JAMIE MATUSOW, JEAN CASSIDY

ASSISTANT EDITOR

JENNIFER VAN WINCKEL

RESEARCH EDITOR

BAHADIR KARUV, PhD

WRITERS IN THIS ISSUE

VJAY CHINTA, LABRO DIMITRIOU, PETER HOLDITCH,
TIM JACOBSON, TRACE LOWE, PRAKASH MALANI,
JOE MITCHKO, ROBERT PATRICK, CHRIS PELTZ,
CLAIRE ROGERS, VADIM ROSENBERG, JIM WEAVER

SUBSCRIPTIONS

For subscriptions and requests for bulk orders,
please send your letters to Subscription Department.

SUBSCRIPTION HOTLINE:

888-303-5282

Cover Price: \$8.99/Issue

Domestic: \$149/YR (12 Issues)

Canada/Mexico: \$169/YR

Overseas: \$179/YR

(U.S. Banks or Money Orders)

PRESIDENT AND CEO

FUAT KIRCAALI

VP, BUSINESS DEVELOPMENT

GRISHA DAVIDA

GROUP PUBLISHER

JEREMY GEELAN

PRODUCTION CONSULTANT

JIM MORGAN

ART DIRECTOR

ALEX BOTERO

LEAD DESIGNER

ABRAHAM ADDO

ASSOCIATE ART DIRECTORS

LOUIS F. CUFFARI • RICHARD SILVERBERG

ASSISTANT ART DIRECTOR

TAMI BEATTY

SENIOR VP, SALES & MARKETING

CARMEN GONZALEZ

VP, SALES & MARKETING

MILES SILVERMAN

ADVERTISING SALES DIRECTOR

ROBYN FORMA

DIRECTOR OF SALES AND MARKETING

MEGAN MUSSA

ADVERTISING SALES MANAGER

ALISA CATALANO

ASSOCIATE SALES MANAGERS

KRISTIN KUHNLE • BETH JONES

PRESIDENT, SYS-CON EVENTS

GRISHA DAVIDA

CONFERENCE MANAGER

LIN GOETZ

FINANCIAL ANALYST

JOAN LAROSE

ACCOUNTS RECEIVABLE

CHARLOTTE LOPEZ

ACCOUNTS PAYABLE

BETTY WHITE

VP, INFORMATION SYSTEMS

ROBERT DIAMOND

WEB DESIGNERS

STEPHEN KILMURRAY • CHRISTOPHER CROCE

ONLINE EDITOR

LIN GOETZ

CIRCULATION SERVICE COORDINATORS

SHELIA DICKERSON • EDNA EARLE RUSSELL

LINDA LIPTON

EDITORIAL OFFICES

SYS-CON Publications, Inc.

135 Chestnut Ridge Road, Montvale, NJ 07645

Telephone: 201-802-3000 Fax: 201-782-9638

SUBSCRIBE@SYS-CON.COM

WebLogic Journal (ISSN# 1535-9581)

is published monthly (12 times a year)

Postmaster: Send Address Changes to

WEBLOGIC JOURNAL, SYS-CON Publications, Inc.

135 Chestnut Ridge Road, Montvale, NJ 07645

732-607-9941 • B.J.G. Associates@cs.com

WORLDWIDE NEWSSTAND DISTRIBUTION

Curtis Circulation Company, New Milford, NJ

NEWSSTAND DISTRIBUTION CONSULTANT

Brian J. Gregory /Gregory Associates/W.R.D.S

732-607-9941 • B.J.G. Associates@cs.com

FOR LIST RENTAL INFORMATION:

Kevin Collopy: 845-731-2684, kevin.collopy@edihroman.com

Frank Cipolla: 845-731-3832, frank.cipolla@epostdirect.com

**SYS-CON
EVENTS**

© COPYRIGHT 2004 BY SYS-CON PUBLICATIONS, INC. ALL RIGHTS RESERVED. NO PART OF THIS PUBLICATION MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING PHOTOCOPYING, RECORDING, OR BY ANY INFORMATION STORAGE AND RETRIEVAL SYSTEM, WITHOUT WRITTEN PERMISSION. FOR PROMOTIONAL REPRINTS, CONTACT REPRINT COORDINATOR, SYS-CON PUBLICATIONS, INC. RESERVES THE RIGHT TO REVISE, REPUBLISH AND AUTHORIZE THE READERS TO USE THE ARTICLES SUBMITTED FOR PUBLICATION. ALL BRAND AND PRODUCT NAMES USED ON THESE PAGES ARE TRADE NAMES. SERVICE MARKS OR TRADEMARKS OF THEIR RESPECTIVE COMPANIES. WLDJ AND WLDJ.COM ARE REGISTERED TRADEMARKS OF SYS-CON MEDIA. WEBLOGIC IS A TRADEMARK OF BEA SYSTEMS. SYS-CON AND WLDJ ARE INDEPENDENT OF BEA SYSTEMS, INC.

CONTACT: joe@sys-con.com

PAGE 4 www.WLDJ.com



Visit us at BEA eWorld – Booth 216

©2003 Wily Technology, Inc. The Wily logo is a trademark of Wily Technology, Inc.

Two years without a vacation.
The application's up. It's down.
It's up. It's down.

I'm to blame. Steve's to blame.
Someone's always to blame.

Not any more.

Get Wily.™

Enterprise
Application Management
1 888 GET WILY
www.wilytech.com



Instrumenting a Java Page Flow Using JMX Technology



BY CHRIS PELTZ &
CLAIRE ROGERS

AUTHOR BIOS...

Chris Peltz is a software architect within HP's Developer Resources organization, providing technical solutions around Web services and adaptive management.

Claire Rogers is a software consultant in HP's Developer Resources organization, providing technical consulting to customers on J2EE application management.

CONTACT...

chris.peltz@hp.com
claire.rogers@hp.com

DESIGNING MANAGEABILITY INTO
YOUR J2EE APPLICATION

With Web services usage on the rise, organizations are seeing a growing complexity in the enterprise systems being built. The need for a robust management solution is critical, as organizations look for better ways to monitor and control their IT environment.



While Gartner has estimated that 40% of unplanned downtime is often caused by application failures, application manageability is often an afterthought for the developer.

The benefits of manageability to an organization are indisputable. With manageability built-in, IT can quickly identify and resolve problems that occur. This can result in increased reliability and a better end-user experience. But, why should the developer care about this? A well-managed application relieves a great amount of burden from the developer. Developers don't have to be woken up in the middle of the night to resolve nonapplication problems. Time typically spent on problem resolution can be focused on developing new application functionality for the business.

To leverage the benefits of manageability, a developer must *design for manageability*. This means carefully considering the approaches and technologies available, including JMX, Logging, ARM, SNMP, and WSDM. The choice is often driven by ease of use, languages and platforms supported, and market adoption. This article focuses on one management standard for Java, JMX.

JMX (Java Management

Extensions) is a specification defining how Java resources can be managed through a common interface. The JMX architecture, illustrated in Figure 1, consists of three layers:

- **Instrumentation layer:** How you create manageable objects in Java.
- **Agent layer:** A set of management agents that control the resources being exposed at the instrumentation layer.
- **Distributed layer:** How to connect the management layer to external applications and protocols (e.g., HTTP, SNMP, or RMI, etc.)

JMX development occurs primarily at the instrumentation layer, where MBeans are used to represent the managed resources. An MBean describes a given management interface through its attributes and operations. MBeans can be developed using either static operations (standard MBeans) or operations that are discovered at runtime (dynamic MBeans). Additionally, JMX offers additional persistence, and caching through Model MBeans.

In this article, we'll demonstrate how JMX MBeans can be developed in BEA WebLogic Workshop, the visual development environment used to develop applications on the BEA WebLogic platform. Workshop has a simplified programming model, based on controls, events, and properties, which greatly enhances the development of J2EE and Web services

application. The IDE enables enterprise applications to be easily built through a robust Model-View-Controller (MVC) architecture.

Case Study

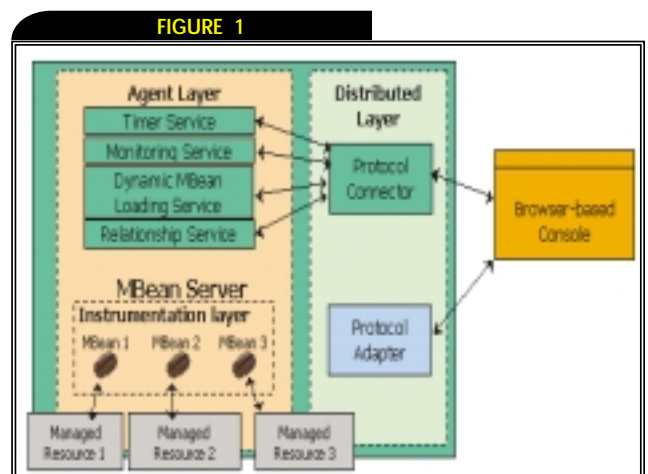
The case study presented here demonstrates an online shopping application called DizzyWorld, which provides key functionality for ordering products from a shopping catalog. Ultimately, we will want to create a business metric that helps line-of-business managers monitor their online business. In order to do this, they need to see the percentage of customers who are visiting the DizzyWorld Web site and completing a purchase. By using JMX, we can keep track of specific business transactions initiated by the customers to help calculate the percentage of purchases completed at any time.

First, to get a better sense of how this application works, let's go through a shopping experience. Initially, the DizzyWorld application displays the main page, which contains different categories of items available from the catalog. We can choose a category to further drill down and view the items in that category. If we select a particular item, we are then shown detailed information on the item and have the ability to add it to our shopping cart. Figure 2 illustrates one step of this process.

Once we click on the Add to Cart button, the items are placed in our shopping cart. We can view the items in our shopping cart by clicking the View Cart link. The next step in the process is to proceed to checkout and place the order. Once we submit the order, the purchasing process is complete.

If we look at the architecture for DizzyWorld, it consists of several J2EE components, including JavaServer Pages (JSPs), Enterprise JavaBeans, and Web services. One BEA WebLogic technology used in our scenario is the Java Page Flow, or JPF. JPF is a Struts-based framework that makes a clean separation between page navigation, business logic, and the data model components. Developers can easily construct JPF flows through visual drag-and-drop tools available in Workshop.

Figure 3 illustrates one of the JPFs in our DizzyWorld application. This flow leverages WebLogic control to easily access certain resources in the application. Essentially, a



JMX architecture



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.



sure the MBean was registered each time the WebLogic Server started.

Now that we have the MBean interface implemented and registered, we can instrument our DizzyWorld application.

Instrumenting Your Code with JMX

With the JMX MBean developed, we can turn our attention to the instrumentation step. Two important development steps must be completed. First, we need to decide how we will discover and invoke the operations on the MBean. We then need to decide where in the JPF we will place our JMX calls to notify our MBean when shopping carts have been created or closed.

The discovery process for an MBean works similar to EJB discovery in a J2EE application. Developers must first locate the MBeanHome and then locate a RemoteMBeanServer instance. At that point, any of the operations on the JMX MBean can be invoked. Listing 5 shows how this can be accomplished for our ShoppingCart MBean.

There are a couple of things to consider in the development. First, you should see references to a MyShoppingCartConstants class. This class, shown in Listing 6, allows us to isolate some of the JMX Server configuration information into a single location. This class contains settings for the BEA WebLogic Server host and login information. We could have also placed this data in an XML configuration file and dynamically loaded it at runtime.

The design should also look for opportunities to increase reuse and reduce complexity. The logic in Listing 5 will become unmanageable if you have to add this code everywhere instrumentation is required. To simplify the business logic, we have developed a Proxy class, MyShoppingCartMBeanHelper, to hide some of the complexity of discovering the JMX MBean and invoking the operations. Listing 7 highlights the interface for this Proxy class. We would place the code shown in Listing 5 in this addCart() method and any clients wishing to invoke this operation can call this method directly.

There is one further refinement we can make to the code. You'll notice that we're doing a lookup of the MBean every time one of the Helper class methods is invoked. We may want to consider caching the reference to the MBean Server once we have obtained it and then use that reference in subsequent invocations. This is where the Home Factory design pattern could be applied to isolate this lookup step.

Now that we have defined a simplified interface for interacting with the MBean, we are ready to instrument the application. This is probably one of the most critical steps in the design, because excessive instrumentation can impact performance and maintainability of the application. To illustrate JMX integration with JPF, we opted to identify events, rather than data elements, that could be instrumented. Referring again to Figure 3, we will define the creation of a shopping cart at the point the user clicks the Proceed to Checkout button, which corresponds to the checkout action. We will also define the completion of a shopping cart as the point where the user clicks the Place Order button, which corresponds to the placeOrder action. The JMX calls will go behind each of these two events.

Before we begin instrumenting the application, we first have to import the class and create an instance of the MyShoppingCartMBeanHelper class we developed. The following code is placed at the top of the CheckoutCartController.jpf class:

```
import com.hp.atc.mbeandemo.*;
private MyShoppingCartMBeanHelper mbeanHelper =
new MyShoppingCartMBeanHelper();
```

At this point, we just have to insert the appropriate JMX calls in our code. From the JPF, we can double-click on the checkout action, which will take us directly to the code. We can then add the following instrumentation line to the end of this routine:

```
mbeanHelper.addCart()
```

We would perform a similar instrumentation step to add a call to closeCart() behind the placeOrder action.

And that's all we have to do to JMX-enable our application. Hopefully, this has shown you how simple it can be to instrument your application with JMX, especially if you leverage design patterns to isolate the JMX logic. But, what if you didn't want to instrument your application? One alternative is to use aspect-oriented programming (AOP) techniques to isolate the instrumentation rules. With AOP, you could define a management aspect that indicates where the JMX calls should be added in the code, without requiring the application code to be directly modified.

We can now turn our attention to testing our instrumented application.

Testing the Instrumentation

The final step in our development is to test the JMX instrumentation. To do that, we will walk through a shopping experience using the application and determine whether the JMX MBean is being updated.

We begin by bringing up a browser and navigating to the home page for the application:

```
http://localhost:7001/OnlineSalesWeb
```

At this point, we can browse through the various items available and place them in our shopping cart. When we are done shopping, we can view our shopping cart by clicking the View Cart button. We would then be presented with a screen similar to Figure 5.

Previously, we added instrumentation at two points in the execution: once when the checkout process was initiated and again when the checkout process was completed. In this case, when we

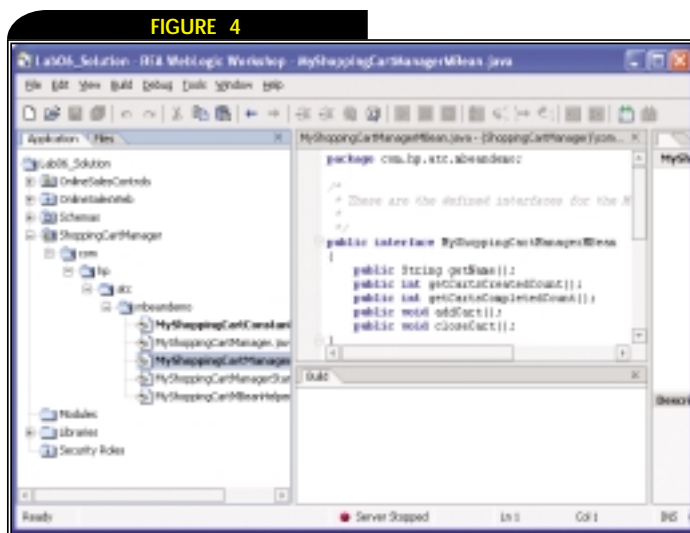


FIGURE 4 JMX Development in WebLogic Workshop

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



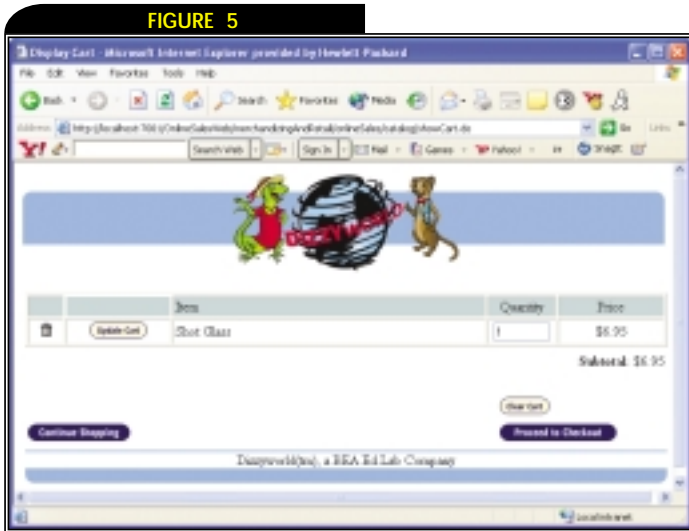
Over 40% of IT development time is spent on manually reconciling and validating the data shared between systems - now there is a better way.

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.

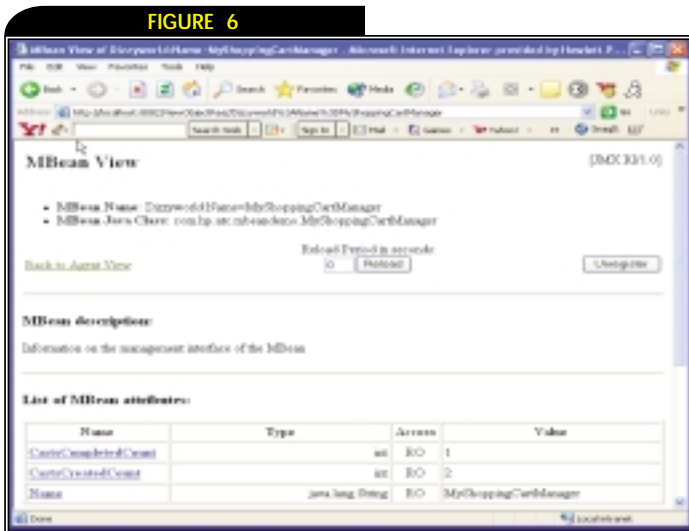
Visit us at eWorld: Booth 331

781-890-2890

www.pantero.com



Shopping cart



ShoppingCart MBean result

click on the Proceed to Checkout button, our JMX MBean should be updated accordingly.

We can use a variety of methods to verify whether the JMX MBean was updated. BEA WebLogic provides a set of command-line tools to return the current values of the JMX MBeans. There are also a number of tools available for browsing JMX data, including EJTools JMX Browser, AdventNet JMX Studio, XMOJO, and XtremeJ. For our purposes, we used a simple HTML adaptor for browsing the MBeans, StartHtmlAdaptor.

Once the WAR file for this Web-based tool is deployed into WebLogic, you can view the deployed JMX MBeans by browsing to the following location:

<http://localhost:8082>

Figure 6 shows one view of our ShoppingCart MBean. As you'll notice, the CartsCreatedCount was updated to two, indicating a new shopping cart was created. At the point when the purchase is approved by the user, you should then see the CartsCompletedCount get updated as well.

Conclusion

Without application manageability, application problems can cost an organization millions of dollars to fix and maintain the software. While it's possible to manage Java applications once in production, we recommend you consider introducing JMX early in the development life cycle. This approach can offer the following benefits:

- **Ease of use:** The JMX specification leverages many of the Java and J2EE concepts, such as JavaBeans and EJBs. Learning JMX is quite easy for the Java developer.
- **Componentization:** With JMX you have the ability to expose as much or as little of your application as you want.
- **Leverage existing management components:** You don't have to get rid of your current management solutions. You can create synergies between what you have today and what you may want to add in the future.

Application manageability allows you to produce robust applications that can adapt to the needs of the business. If you take time to add manageability, you improve the time-to-market because problems in the application can be found during development or testing, and resolving problems once in production can be done quicker. And, as this article has shown, you can apply JMX technology to solve real business problems, such as determining shopping cart success rates for a business.

In this article, we demonstrated how a JPF could be instrumental with JMX. Hopefully, you have a better understanding on how to get started using JMX and building manageability into your Java application with the BEA WebLogic Platform. In a future article, we will look at taking this JMX-enabled application and integrating it into HP OpenView Operations.

References

- **Murray, Justin.** "Enhancing Application Manageability". *WebLogic Developer's Journal* (www.sys-con.com/weblogic) Volume 2, issues 6 and 7.
- **Davidson, Stephen and Rogers Claire.** "Instrument Your Application Using JMX Tutorial": http://devresource.hp.com/drc/tutorials/JMX/mBean_123_tutorial.jsp.

Listing 1

```
package com.hp.atc.mbeandemo;

public interface MyShoppingCartManagerMBean
{
    public String getName();
    public int getCartsCreatedCount();
    public int getCartsCompletedCount();
    public void addCart();
    public void closeCart();
}
```

Listing 2

```
package com.hp.atc.mbeandemo;

public class MyShoppingCartManager implements
MyShoppingCartManagerMBean, java.io.Serializable
{
    private int cartsCompleted = 0;
    private int cartsCreated = 0;

    public MyShoppingCartManager() {
    }

    public String getName() {
        return "MyShoppingCartManager";
    }
}
```




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



Are your Web applications open 24 hours? Join Sun's expert John VanSant and H&W for a Web seminar series April 6 and June 22 to learn how to make your J2EE applications run at top speed. Register today at: www.hwcs.com/wldj2.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.

See us at BEA eWorld booth #330

© 2003-2004 H&W Computer Systems, Inc.
DiagnoSys is a trademark of H&W Computer Systems, Inc.



Bridging the Enterprise Computing Gap For 25 Years

```

public int getCartsCreatedCount() {
    return cartsCreated;
}

public int getCartsCompletedCount() {
    return cartsCompleted;
}

public void addCart() {
    cartsCreated++;
}

public void closeCart() {
    cartsCompleted++;
}
}

```

Listing 3

```

package com.hp.atc.mbeandemo;

import weblogic.common.*;
import java.util.*;
import javax.management.*;
import weblogic.management.*;

public class MyShoppingCartManagerStartup implements T3StartupDef
{
    private T3ServicesDef services;
    MyShoppingCartManager mbean = null;

    public MyShoppingCartManagerStartup() {
        System.out.println("Creating MyShoppingCartManagerStartup
class");
    }

    public String startup (String name, java.util.Hashtable args)
        throws Exception {
        String serverName = (String)args.get("ServerName");
        String mbeanName = (String)args.get("MBeanName");

        System.out.println("Instantiating and registering " + mbeanName);
        mbean = new MyShoppingCartManager();

        this.registerMBean((Object) mbean,mbeanName);
        return mbeanName + " registered with the MBean server";
    }

    public void setServices(T3ServicesDef services) {
        this.services = services;
    }

    private void registerMBean(Object mbeanObj, String mbeanName)
    {
        ObjectName objName = null;

        try {
            objName = new ObjectName(mbeanName);
        } catch (javax.management.MalformedObjectNameException mone) {
            System.out.println("MalformedObjectNameException: " + mone);
        }

        // Register the custom MBean with the MBeanServer.
        try {
            MBeanHome
home=Helper.getMBeanHome(MyShoppingCartConstants.username,
MyShoppingCartConstants.password,
MyShoppingCartConstants.wlServerHost,
MyShoppingCartConstants.wlServerName);
RemoteMBeanServer rmbs = home.getMBeanServer();

            rmbs.registerMBean((Object)mbeanObj, objName);
            System.out.println("\n["+mbeanName+"] MBean registered
...");

        } catch(InstanceAlreadyExistsException i) {
            System.out.println("MBean (" + objName + ") already
exists");
        } catch(javax.management.MBeanRegistrationException mbre) {
            System.out.println("MBeanRegistrationException: " + mbre);
        } catch(javax.management.NotCompliantMBeanException ncmbe) {

```

```

        System.out.println("MBeanRegistrationException: " + ncmbe);
    }
}

```

Listing 4

```

<StartupClass
Arguments="ServerName=dizzyAdmin,MBeanName=Dizzyworld::Name=MyShoppingCartM
anager"
    ClassName="com.hp.atc.mbeandemo.MyShoppingCartManagerStartup"
    FailureIsFatal="true" Name="MyShoppingCartStartup"
Targets="dizzyAdmin"/>
<Application Name="StartHtmlAdaptor"
Path="C:\student\course_plat_develop\domains\dizzyworld\dizzyAdmin\upload"
    StagingMode="nostage" TwoPhase="true">
    <WebAppComponent Name="StartHtmlAdaptor" Targets="dizzyAdmin"
URI="StartHtmlAdaptor.war"/>
    </Application>
</Domain>

```

Listing 5

```

// Code to invoke the ShoppingCart addCart() operation.
ObjectName objName = null;
try {
    objName = new ObjectName(MyShoppingCartConstants.mbeanName);
}
catch (javax.management.MalformedObjectNameException mone) {
    System.out.println("MalformedObjectNameException: " + mone);
}
try {
    MBeanHome home=Helper.getMBeanHome(MyShoppingCartConstants.username,
MyShoppingCartConstants.password,
MyShoppingCartConstants.wlServerHost,
MyShoppingCartConstants.wlServerName);
RemoteMBeanServer rmbs = home.getMBeanServer();
if (rmbs != null)
    rmbs.invoke(objName,"addCart",null,null);
} catch (InstanceNotFoundException inf) {
    System.out.println("InstanceNotFoundException caught: " +
inf);
} catch (MBeanException mbe) {
    System.out.println("MBeanException caught: " + mbe);
} catch (ReflectionException re) {
    System.out.println("ReflectionException caught: " + re);
}
}
}

```

Listing 6

```

package com.hp.atc.mbeandemo;

public class MyShoppingCartConstants
{
    // This will be the name of the MBean being registered.
    public static String mbeanName =
"workshop::Name=MyShoppingCartManager";

    // This will be the name of the WebLogic server instance and host/port
    public static String wlServerName = "cgServer";
    public static String wlServerHost = "t3://localhost:7001";

    // This is the username and password associated with the admin for
registration
    public static String username = "weblogic";
    public static String password = "weblogic";
}

```

Listing 7

```

import javax.management.*;
import weblogic.management.*;

public class MyShoppingCartMBeanHelper implements
MyShoppingCartManagerMBean, java.io.Serializable
{
    public void addCart() {
        // place code from Listing 5 here.
    }

    public void closeCart() {
    }

    public int getCartsCreatedCount() {
    }

    public int getCartsCompletedCount() {
    }
}

```


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.



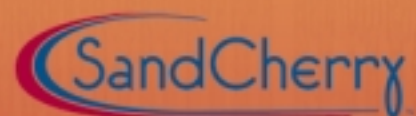
For additional information:

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

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

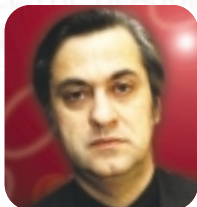
Info@sandcherry.com
www.sandcherry.com

Download
30 Day Free Trial
www.sandcherry.com



An Architectural Blueprint

PART 2



BY LABRO DIMITRIOU

AUTHOR BIO

Labro Dimitriou is a BPMS subject matter expert and grid computing advisor. He has been in the field of distributed computing, applied mathematics, and operations research for over 20 years and has developed commercial software for trading, engineering, and geo-science. Labro has spent the last five years designing BPM-based business solutions.

CONTACT...

labro@bpmsinc.com

THE REASONS FOR BUILDING A MODEL

Let's dive into the murky waters of modeling, describe some of its challenges, and provide, an overview of the state of business process modeling.

In my first article in this series (*WLDJ*, Vol. 3, issue 4), I discussed the importance of architectural blueprints and best practices in order to establish repeatable ways for building robust, enterprise-wide integration solutions, for an adaptable and agile enterprise. I then established that service is the unifying construct that merges SOA and BPMS with Web services, as the underpinning of connectivity, in highly distributed and ever-changing business ecosystems. SOA is an evolutionary step in distributed computing where business process and BPMS is an evolutionary new technology innovation – a first-class citizen in computing. I concluded with the notion of elementary business services (EBS) – small units of work made available to the enter-

prise via the enterprise info bus (yet another anachronistic and overloaded term). The portfolio of EBS delivers the ultimate guide to enterprise-level reuse. New business processes are orchestrated in near real time by aligning existing EBS, new business events, and human resources with adaptive corporate objectives.

In this article I discuss emerging business process design patterns that provide BPM-centric architectural solutions to long-standing, enterprise-wide business challenges. After a look at what it takes to model, we'll deconstruct business processes to components we know well and try to understand the surrounding design challenges. Finally, I will provide a proposed taxonomy for business-process design patterns.

Meanwhile, the board of the fictive Car Insurance Agency whose business process get car insurance quote I was going to model asked me to wait a bit before I presented the solution. They are going through a business

process redesign phase and they are about to approve a new policy. Under the new policy all quotes have to be authored by an external underwriter. The underwriters come in via a pool of a virtual collaborative B2B exchange. Additionally, they are negotiating, as part of the process, a credit report check via secure Web services. Therefore, despite what I promised in my first article, and for the sake of completeness, I will have to defer the discussion for the next article.

The Problem with Modeling

Booch et al tells us “a Model is a simplification of reality” and that “the best models are connected to reality”; but whose reality are we modeling? Furthermore, what kind of modeling paradigm or framework do we use to model the modeling framework? And why do we need models? They can be dangerous because they make you forget reality. Consider the following analogy: business domains are like dreams and models are what you answer when somebody asks you to explain them. Pretty soon you remember your answers and forget the dreams. To that end, my objective is not to answer exhaustively all these matters, even if I could. It is merely to bring awareness of the underlining concepts and provide a brief account of generally accepted notions and challenges involved in (1) modeling business ecosystems with processes – business processes; (2) using modeling frameworks that model business processes; (3) selecting a language/syntax to present the modeling framework; and (4) selecting a graphic notation to visually render business processes.

Business processes model collaborative business ecosystems well and the BPMS framework successfully bridges the impedance mismatch between business and IT. But whose modeling standards should you use? BPMN has been around for a while, but BPEL4WS seems to be the winner; although by the sheer weight of the proverbial 900-pound IT gorilla(s). Clearly, both standards use XML as the implementation choice. On the other hand, the UML camp is not standing still. Are UML sequence diagrams good enough for modeling? What is this buzz about OMG’s model-driven architecture, not to be confused with aspect-driven architecture or domain-driven design? Not to mention, of course, numerous other standards such as XPDL, ebXML, XLANG, WSCI, WSFL, and many others brewing in the academic and research world, such as YAWL (yet another workflow language). Now is a good time to break a common fallacy: contrary to popular belief, workflow and process do share very similar aspects. That was true well before workflow software companies hijacked the term workflow to mean document flow and work allocation, and BPM evangelists, including myself, wanted nothing to do with things of the past like <e>AI and workflow engines. (I use the <e> instead of the traditional E to denote AI beyond the firewall and across business ecosystems.) Clearly, graphical, control-flow representation, and graph theory are the common aspect of workflow and process alike. So from now on, I’ll use the terms workflow and process interchangeably.

Business-process models don’t encapsulate business domain-specific knowledge but have expressive power in defining business content. On the other hand, this opacity of business content has a non-deterministic effect on the business protocol, making exception handling and compensating transactions more challenging.

Two mathematical/modeling theories are primarily used to model processes: (1) Petri nets, and (2) π -calculus or variances and supplemental notions such state charts, timed stochastic nets, and ambient calculus, respectively.

Petri nets were introduced by C. A. Petri in the early 1960s as a mathematical tool for modeling distributed systems and, in particular, notions of concurrency, nondeterminism, communication, and synchronization. π -calculus was defined by Milner, Parrow, and Walker as “Calculus of Mobile Processes.” Petri nets-based languages perform better in state-based workflows but have difficulty and increased complexity in modeling multiple concurrent processes and complex synchronization requirements.

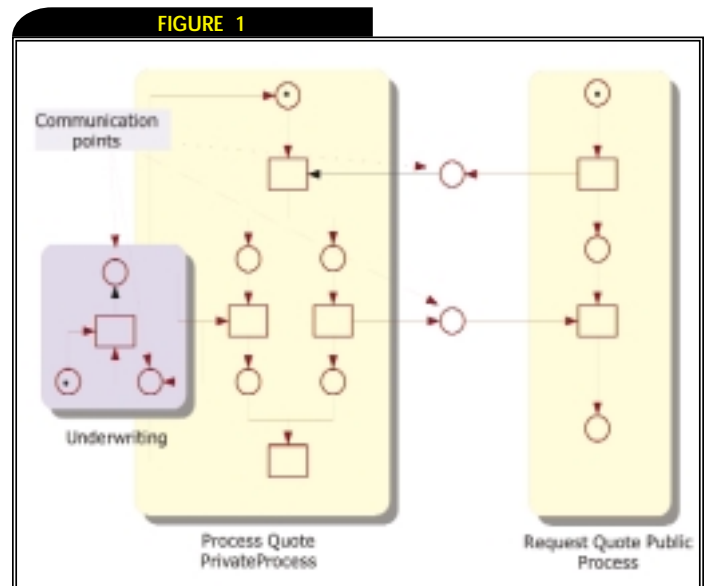
Simply put, graphs have nodes that are connected with edges. Petri net is a kind of graph with nodes that are places (circles) or transitions (rectangles) and tokens. Nodes of different kinds are connected together by means of arcs. There are two kinds of arcs: input and output. A place can hold one or more tokens. The state of a process is modeled by places and tokens and state transitions are modeled by transitions. Figure 1 demonstrates a B2C and B2B interaction as Petri nets, a client requesting an insurance quote from an agent and underwriting process, respectively. The private processes operate essentially independently, but have to synchronize via the shared points. Professor Wil van der Aalst’s presentation provides a simple introduction to Petri nets with a number of good examples and an applet to design your own Petri nets.

π -calculus models concurrent processes communicating via channels where the network topology can change dynamically. Nodes are processes and edges are named channels. Processes can exchange names over channels and there are no other values than channel names. For example, the notation $x(y).P$ means that a process P receives a value y over a channel x. $P1|P2$ indicates that P1 and P2 are two concurrent processes. Finally, (y) means to send a name y over a channel a. For example, the process of a user requesting a quote via an insurance Web site would like something like this:

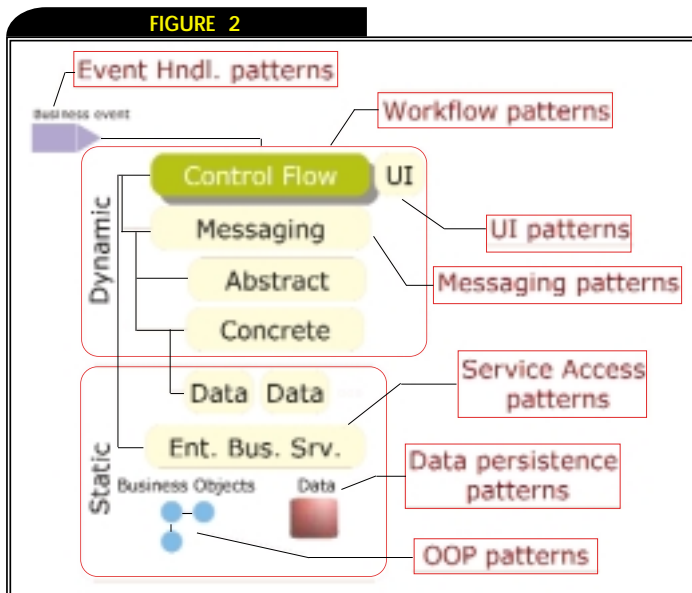
```
webChannel(sendData).RequestQuote | webChannel (getData).ProcessQuote,
```

where RequestQuote and ProcessQuote are two processes running in parallel.

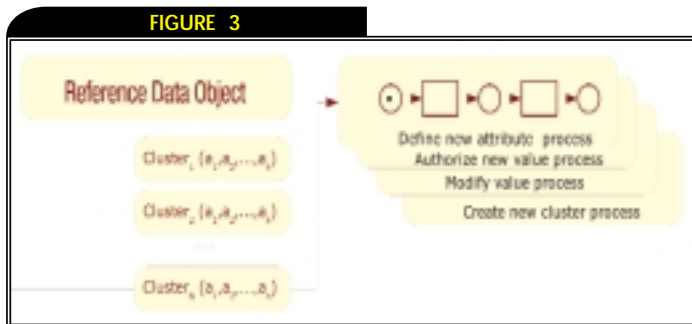
Why complicate things with such formalism you may ask. Because otherwise it would be equivalent to doing accounting without dual book entry techniques and general ledgers, and hop-



Petri net interaction



Layered view of business process and associated design patterns



Reference data

ing that it would all be a zero sum at the end. The underlying process algebra helps us (BPM engines and next-generation business activity monitors) to find deadlocks and race conditions, reduce processes to simpler ones, and find optimal paths, better opportunities (based on business rules), and answer other interesting questions. The vision of BPMS goes well beyond simple execution of processes. It is exactly because of BPMS' formalism that we can now have a direct real-time API to the runtime enterprise and achieve executive dashboard nirvana.

In terms of graphical notation, Business Process Modeling Notation (BPMN) seems the only game in town outside academic and research campuses. While there are talks about implementing BPMN on top of BPEL4WS, a few vendors have announced compliance, including Popkin Software, a software analysis tool vendor; and Intalio, a pure play BPM tool. BEA's WebLogic Platform 8.1 has taken the route of an all-encompassing IDE for designing and deploying distributed applications, based on WebLogic Server, the de facto app server industry standard. The IDE uses a few powerful and intuitive constructs to facilitate business-process design and development. The visual paradigm successfully hides the rigors of OO programming, J2EE, J2EE CA, JMS, and WS-WSDL from the unwary. Most other vendors use typical Visio-like workflow or object-oriented UML notation.

Deconstructing Business Processes

The business process management approach is essentially a

top-down approach. Starting at the top, the breadth of a process aligns all knowledge domains within the agile enterprise: business intelligence, subject matter expertise interaction (UI and other), location and organizational boundaries, legacy applications, integration points, and data requirements. As we increase the depth of the process we identify subprocesses – some within the boundaries of a line of business – down to the micro level of individual business rules. The layered top-down approach makes BPMS the perfect vehicle for legacy retiring. Enterprise-wide processes can trigger and execute subprocesses and so on. Clearly this top-down approach facilitates incremental change rather than “a big bang” approach.

Gregor Hohpe and Bobby Woolf, in *Enterprise Integration Patterns*, conclude:

The business process component unites a series of services into logical unit that interacts with other such units via messaging to achieve highly scalable, resilient flows of logic and data. The coalescence of process, object, and interaction patterns into business process component is the future.

Figure 2 provides a layered view of a business process and the associated design patterns involved at each layer. A business process starts with a business event triggered by an actor, real person (internal or external to the organization), or a system. A business process contains dynamic and static aspects.

The dynamic aspect is designed using drag-and-drop mechanisms within an IDE and encapsulated by the control flow language and messaging/connection points, including <e>AI, legacy adapters, and Web services. The control flow soft wires volatility, making it easy for domain experts and design modelers to make changes and deploy them with a click of a button. Messages are implemented using abstract and concrete classes. Concrete classes hide the protocol-dependent implementation details from the control flow.

The static layer contains the usual suspects: business services, business objects, and data. There are two places for data in a business process: within the exchange of messages for protocol and business content, and the bottom of the stack for persistence and other business-process metadata repositories. To restrain any RDBMS modeling ideas, I recall Ian Graham's view of data from *Object-Oriented Methods: Principles and Practice*: “It is my firm conviction that data driven methods are dangerous in the hands of someone educated or experienced in the relational tradition.”

BPEL4WS discusses messaging only in light of Web services. Contrary to that, BEA's WebLogic Platform 8.1 neatly enables encapsulation of any imaginable entity as a control and exposes method calls that can be used as connection points. Through introspection, it can even expose internal methods as “straight-through” connection points. I'll talk more about the power of controls on the next article.

Business Processes Design Patterns

Clearly, the dynamic layer of a business process establishes the foundation for business-process patterns: combination of workflow and Web services/messaging patterns. In addition, as a new breed of OEMs, merging subject matter expertise in specific business verticals with technology, start rolling out portfolios or libraries of highly configurable executable business processes, we will witness the emergence of policy patterns or best practices business processes.

-continued on page 50

BEA WebLogic 8.1™ is built for SOA Performance.

Acsera is built to guarantee it.

Application Monitoring and Performance Management for BEA WebLogic 8.1.

Acsera resolves your application performance problems in 60 minutes.
And it will change the way you manage performance on BEA WebLogic 8.1.

See the demo at www.acsera.com/demo

It takes just 15 minutes. You have plenty of questions about the challenges of managing complex applications on service-oriented architecture. We have the answers. And your BEA WebLogic 8.1 applications will run like never before. See the live demo at Booth #517

ACSeRA

Application Monitoring and Performance
Management for BEA WebLogic 8.1.

Considering MySQL?

Read On...



PART 2

A POWERFUL COMBINATION FOR MISSION-CRITICAL APPLICATIONS



BY PRAKASH MALANI

AUTHOR BIO...

Prakash Malani has extensive experience in architecting, designing, and developing object-oriented software, and has done software development in application domains such as entertainment, retail, medicine, communications, and interactive television. He practices and mentors leading technologies such as J2EE, UML, and XML. Prakash has published various articles in industry-leading publications.

CONTACT...

pmalani@ebuilt.com

This article explores using MySQL as the database engine where the application is developed using BEA WebLogic Workshop 8.1 and deployed to BEA WebLogic Server 8.1. Using an archetypical J2EE architecture, I evaluate the impact of using MySQL from various aspects such as choosing the correct version of MySQL, setting-up the server, and making development adjustments. The information presented here not only enhances the readers' understanding of the tools and technologies utilized, but also saves countless hours. Even readers who employ different database technologies will find the information and material useful.

Last month (*WLDJ* Vol. 3, issue 4), I discussed how to select the "right" version of MySQL and described various changes to the WebLogic Domain Configuration to support key J2EE technologies such as Java DataBase Connectivity (JDBC) and Java Message Service (JMS). This month, I'll describe the changes, adjustments, and modifications to support Enterprise JavaBeans (EJBs), the core component model for J2EE as well as Java Transaction API (JTA). Just as

in Part I, the development tool of choice is WebLogic Workshop 8.1 and the application is deployed to WebLogic Server 8.1.

Subsystem Architecture

A J2EE application consists of many subsystems or modules. The archetypical application consists of the following subsystems: UtilProject, DomainProject, AgentProject, TestProject, and SEMAppWeb. These subsystems, as well as dependencies between them, are depicted in Figure 1. The UtilProject contains common utility classes. The DomainProject contains the container-managed persistence (CMP) EntityBeans as well as a Session Facade Stateless Session Bean (SLSB). The AgentProject contains a SLSB that enqueues messages on to a queue for asynchronous processing, and a Message-Driven Bean (MDB) to process the messages. The TestProject aggregates all the tests from other projects. The SEMAppWeb project is the default web project.

Before proceeding further you need to complete the following steps:

- Create a BEA WebLogic Workshop application (e.g. SEMApp).
- Choose SEMDomain as the server.
- Add junit.jar into the library.

- Import a Java project within SEMApp named UtilProject.
- Import an EJB project within SEMApp named DomainProject.
- Import an EJB project within SEMApp named AgentProject.
- Import a Java project within SEMApp named TestProject.
- In accordance with the project dependencies, change the build order to the following: UtilProject, DomainProject, AgentProject, SEMAppWeb, and TestProject.
- Build all the projects.
- Start WebLogic Server.

The next section starts with a high-level overview of the Enterprise JavaBeans and specifically explores the issues that arise when using them with MySQL.

Enterprise JavaBeans (EJBs)

Enterprise JavaBeans are the core component model for J2EE. There are three types of EJBs: Session Beans, Entity Beans, and Message-Driven Beans. Entity Beans are specifically designed to interact with the database. There are two types of Entity Beans: Entity Beans with bean managed persistence (BMP) and Entity Beans with container managed persistence (CMP). When using BMP, the programmer writes all the JDBC code and has control over that code. When using CMP, the programmer declaratively specifies persistence in eXtensible Markup Language (XML). The EJB container is responsible for providing the persistence by generating the necessary JDBC code. The archetypical J2EE blueprint architecture uses Entity Beans with CMP.

The following sections discuss issues that arise when using MySQL and Entity Beans with CMP. One issue is automatic primary key generation. The other issue is deferrable constraints. The archetypical sample application consists of many CMP entity beans to illustrate these issues. Figure 2 shows a class diagram of the CMP entity beans and the relationships between them.

Automatic Primary Key Generation

BEA WebLogic Server supports two methods for automatic primary key generation. The first method, known as the NamedSequenceTable strategy, uses a sequence table. This approach is generic and works with most databases, including MySQL. As an example, for the Person table, create a Person_Seq table for primary key generation. The Person_Seq table has one column named Sequence. Insert a row into Person_Seq where the sequence starts, typically zero. Refer to sem.sql in the provided source code example. Insert the following EJBGen tag for automatic primary key generation for each entity.

```
*
* @ejbgen:automatic-key-generation
*   cache-size="10"
*   name="Person_Seq"
*   type="NamedSequenceTable"
*
```

Refer to the source code for PersonBean.ejb in the example source code (the source code is online at www.sys-con.com/weblogic/sourcecec.cfm).

The second method takes advantage of native DBMS support. For example, Oracle has sequences that can be utilized for automatic primary key generation whereas Microsoft SQL Server has identity columns for automatic primary key generation. BEA WebLogic Server supports the native DBMS feature for Oracle and Microsoft SQL Server databases only.

Using a single, automatic, primary key generation technique throughout the project is recommended. Since the strategy of NamedSequenceTable is officially supported, that strategy is recommended. Further information about automatic primary key generation support is available at <http://e-docs.bea.com/wls/docs81/ejb/entity.html#1155399>.

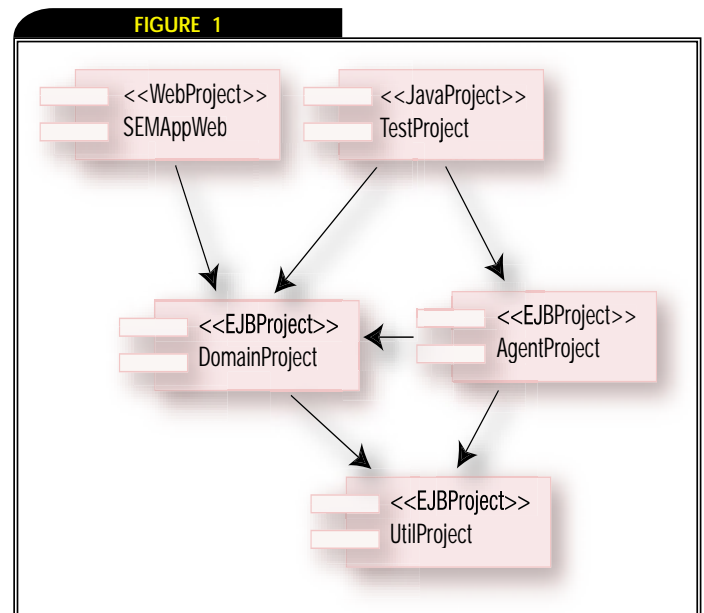
Deferrable Constraints

Databases such as Oracle and DB2 support a feature known as deferrable constraints. When using a deferrable constraint, the constraint, such as a foreign key, is not checked right away during an insert or update. The constraint is checked later at the commit time. This capability gives the WebLogic Server CMP engine considerable leeway in ordering SQL statements for maximum efficiency and performance. However, MySQL does not support deferrable constraints. Therefore, the default CMP behavior will cause foreign key violations as the constraints are always checked right away. For example, the relationship between Person and PersonStatus entities demonstrates the problem. A potential solution is to turn off the ordering of database operations. Insert the following EJBGen tag to turn off the ordering of database operations:

```
* @ejbgen:entity
*   ...
*   order-database-operations="false"
```

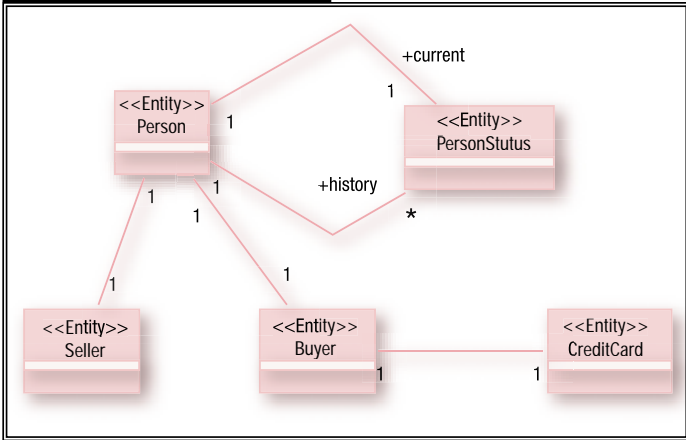
However, this tag has no effect whatsoever. By default, batching of statements is enabled. If batching of statements is enabled, then ordering of database operations is automatically enabled. Therefore, turn off both the batching of operations and ordering of database operations. Insert the following EJBGen tag to turn off ordering of database operations and batching of statements.

```
* @ejbgen:entity
*   ...
*   order-database-operations="false"
*   enable-batch-operations="false"
```



Subsystem architecture diagram

FIGURE 2



CMP Entity Beans And Relationships Between Them

Refer to the source code for PersonBean.ejb in the code example. Specifying order-database-operations and enable-batch-operations to “false” is recommended for all the entity beans in the project.

Choosing Date and Time Column Types

Date and time types are used in many different ways in an application. A specific example where date and time are used is to maintain audit information such as create date and update date. The create date records the date and time when the row is inserted. The update date records the date and time when the row is modified. There are a couple of different ways to approach and solve this puzzle. One approach is to maintain and update all the information programmatically in code. The advantage of this approach is that the programmer has complete control over the code. The disadvantage of this approach is that code has to be written, tested, debugged, and maintained.

Another approach is to leverage database-specific features that will automatically insert or update the information. The advantage of this approach is that the programmer does not have to write, test, debug, and maintain the code. The disadvantage of this approach is that the programmer is at the mercy of the facilities available in the database. One way to automatically insert and update dates is to leverage database triggers. Unfortunately, the current version of MySQL does not support these, although a future major version will support them. However, MySQL can still be leveraged to automatically insert and update date and time types. Consider the date and time column types available in MySQL: DATE, TIME, DATETIME, TIMESTAMP, and YEAR. For the auditing purposes, as date and time are both important only DATETIME and TIMESTAMP make sense.

Consider the DDL for the BUYER table. Both the CREATE_DATE and UPDATE_DATE columns are DATETIME column types. The programmer is responsible for maintaining CREATE_DATE and UPDATE_DATE. One recommended place to set the create date is in the ejbCreate() method. Refer to the BuyerBean.ejb included in the example. A recommended place to set the update date is in the session facade update method. Refer to the updateBuyer() method in DomainFacadeBean.ejb in the example.

Consider the DDL for the PERSON table. Both the CREATE_DATE and UPDATE_DATE columns are TIMESTAMP column types. Updating a row in the PERSON table automatically updates the CREATE_DATE column whereas the UPDATE_DATE

column remains unchanged. This result is unexpected! There are at least two potential ways to get the expected result. One is to change the order of the CREATE_DATE and UPDATE_DATE columns as illustrated by the ADDRESS table. The other is to change the column type of CREATE_DATE to DATETIME and set the CREATE_DATE column programmatically as illustrated by the PRODUCT table.

Consider the DDL for the SELLER table. Both the CREATE_DATE and UPDATE_DATE columns are TIMESTAMP column types. However, compared with the PERSON table, the order of CREATE_DATE and UPDATE_DATE is reversed as listed in the DDL. The column UPDATE_DATE is listed and created before the CREATE_DATE column. Updating a row in the SELLER table automatically updates the UPDATE_DATE column, leaving the CREATE_DATE column intact. This is the expected result.

Consider the DDL for the CREDIT_CARD table. The CREATE_DATE column type is DATETIME whereas the UPDATE_DATE column type is TIMESTAMP. The CREATE_DATE column is set programmatically. MySQL updates the UPDATE_DATE column automatically when the row is updated in the table. This is the expected result. One recommended place to set the create date is in the ejbCreate() method. Refer to the code for CreditCardBean.ejb included in the source code example.

Remember to keep the following rules in mind about the TIMESTAMP column type:

- The column value is automatically set to the current date and time, if NULL is inserted into any TIMESTAMP column.
- The column value is automatically set to the current date and time upon create or update for only the first TIMESTAMP in the row column if no value is inserted into that column.
- Inserting an explicit date and time value explicitly defeats time-stamping.

Carefully choosing one policy and implementing it throughout the complete project is recommended.

Other Recommendations

One recommendation is to always use object types rather than primitive types. For example, use java.lang.Integer instead of int. The default value for java.lang.Integer is null where as the default value for int is zero. If there is a nullable foreign key constraint, then the constraint is violated when using the default value of zero because no such matching row exists in the foreign table.

Many properties can be set to enable outputting of more information to assist in debugging and monitoring. weblogic.ejb20.cmp.rdbms.codegen.verbose is one such property. Setting the property to true displays the JDBC statements as well as binding of parameters to the statements.

JTA Domain Configuration Changes

JTA is an application programming interface (API) used to coordinate distributed transactions between different resources. For example, using JTA, a JMS message can be sent or received and data can be committed to the database via JDBC within a single global transaction. In this example, one resource is the JMS server and another resource is the RDBMS. The distributed transaction spans the JMS server as well as the RDBMS. In order to support distributed transactions, resources such as database servers or JMS servers need to support industry standard X/Open XA protocol. The drivers that access the resources need to support XA as well.



How Can You Sink Your Application Overhead?

With Cyanea/One®, your company can ensure that the resource consumption of its J2EE applications stays below desired thresholds. Through Cyanea/One's Performance Analysis and Reporting (PAR) engine, you can quickly baseline an application and trend its performance and resource utilization over time. Automatically identify hotspots within the code and take corrective action. Be instantly notified when key threshold conditions are approaching. Improve service levels and customer satisfaction. Accomplish all these without touching your code or even requiring knowledge of your application.

...Keeping **Your Applications Under PAR.**



Visit us at BEA eWorld 2004 - Booth 235

Find PAR at:
www.cyanea.com/wldj/underpar.html
1-877-CYANEA8 | sales@cyanea.com

Cyanea® and Cyanea/One® are registered trademarks of Cyanea Systems Corp.

Consider for example the `AgentFacadeBean`'s `enqueuePerson()` method. The method not only writes to the database, but also enqueues a message onto a queue. Both the JMS as well as the RDBMS resources need to be XA enabled.

If the JMS resource is not XA enabled, then obtaining the connection factory using a resource reference results in an error. The error message informs you that "two-phase commit is not available". For the `SEMDomain`, set the property on the connection factory to enable XA.

Enabling XA on the JMS Connection Factory

The following steps describe enabling XA on the JMS connection factory.

1. Make sure BEA WebLogic Server is running.
2. Launch the WebLogic Server Console.
3. Log into the Console.
4. Select Services / JMS / Connection Factories.
5. Select `semJMSConnectionFactory`.
6. Select Transactions tab.
7. Enable XA Connection Factory Enabled check box.

The JDBC datasource is not XA enabled. Neither MySQL nor MySQL's JDBC driver support XA. Executing the `AgentFacadeBean`'s `enqueuePerson()` method results in `TransactionRolledbackException` complaining that the "JDBC driver does not support XA", hence cannot be a participant in two-phase commit. To force this participation, set the `EnableTwoPhaseCommit` property on the corresponding `JDBCTxDataSource` property, to true."

BEA WebLogic Server allows distributed transactions even if at the most one of the resources does not support XA. It has the capability to emulate XA for the resource that does not support XA. To emulate XA and allow MySQL to participate in two-phase commits, modify the datasource configuration that accesses MySQL to emulate XA.

Emulating XA

The following steps describe changing the datasource configuration to emulate XA.

1. Make sure BEA WebLogic Server is running.
2. Launch the WebLogic Server Console.
3. Log into the Console.
4. Select Services / JDBC / Data Sources.
5. Select a JDBC Data Source (e.g. `semJDBCDataSource`).
6. Display Advanced Options by clicking on Show.
7. Select Emulate Two-Phase Commit for non-XA driver checkbox.

Transactions and Redelivery

In the sample application, the `ProcessPersonBean` Message-Driven Bean (MDB) listens to the queue and processes messages off the queue. The `ProcessPersonBean` is configured with Container Managed Transaction Demarcation (CMTD) with the transaction attribute of Required. The `onMessage()` method executes within a container transaction. Calling the `setRollbackOnly()` method of the `MessageDrivenContext` rolls back the transaction. WebLogic Server also rolls back the transaction in the event of a runtime exception thrown from the `onMessage()`. Throwing runtime exceptions from the `onMessage()` method is considered a programming error according to the JMS specification.

If the transaction is rolled back, WebLogic Server redelivers the message. The default redelivery count value is -1, meaning there is no limit to the number of times the message is redelivered. No limit causes BEA WebLogic Server to continuously reprocess the message over and over again as long as the transaction is rolled back. The WebLogic Server server can be thrown into an infinite loop. To stop WebLogic Server from endlessly processing the same message over and over again, the redelivery count value can be modified. For the `SEMDomain`, configure the redelivery count value to 0 or higher. The redelivery count value is modified differently based on whether the destination is a regular (i.e., nondistributed) or a distributed destination. One way to modify the redelivery count for a nondistributed destination, is to modify the redelivery count on the destination itself. One way to modify the redelivery count for a distributed destination is to modify the redelivery count on the JMS template corresponding to the distributed destination.

Setting the Redelivery Count Value

The following steps describe changing the redelivery count value for the `PersonQueue`.

1. Make sure WebLogic Server is running.
2. Launch the WebLogic Server Console.
3. Log into the Console.
4. Select Services / JMS / Templates.
5. Select `PersonQueue`.
6. Select Configuration tab.
7. Select Redelivery sub tab.
8. Change the value of Redelivery Limit to 3.

FREE* CD! (\$198.00 VALUE!)

Secrets of the Java Masters

Every *JDJ* Article on One CD!



— The Complete Works —

CD is edited by *JDJ* Editor-in-Chief Alan Williamson and organized into 33 chapters containing more than 1500 exclusive *JDJ* articles!

All in an easy-to-navigate HTML format! **BONUS: Full source code included!**

ORDER AT WWW.SYS-CON.COM/FREECD

*PLUS \$9.95 SHIPPING AND PROCESSING (U.S. ONLY)

©COPYRIGHT 2004 SYS-CON MEDIA. WHILE SUPPLIES LAST. OFFER SUBJECT TO CHANGE WITHOUT NOTICE. ALL BRAND AND PRODUCT NAMES ARE TRADE NAMES, SERVICE MARKS OR TRADEMARKS OF THEIR RESPECTIVE COMPANIES.



Only from the World's Leading i-Technology Publisher

Running and Verifying

To verify that everything is configured properly and the application is deployed, run the MasterTest class. The MasterTest aggregates all the JUnit tests. Copy sample-build.properties to build.properties and change the property values to match the environment. Run the MasterTest by invoking ant as follows: ant invokeMasterTest.

Conclusion

This article discussed issues such as primary key generation and deferrable constraints that arise when using Entity Beans with container-managed persistence and MySQL. I described changes to the WebLogic Domain Configuration to support the Java Transaction API. Knowing and understanding the impact of using MySQL with various J2EE technologies such as EJBs, JMS, JDBC, and JTA is a must for successful project implementation. As illustrated, MySQL, BEA WebLogic Workshop, and BEA WebLogic Server form a powerful combination to architect, design, and deploy mission critical applications.

Acknowledgments

I want to thank Steve Ditlinger, Roshni Malani, and Sarah Woo for reviewing this article and providing invaluable feedback.

References

- Malani, Prakash. "Considering MySQL? Read On..., part I". *BEA WebLogic Developer's Journal*, Vol. 3, issue 4
- To discuss the article and ask questions start here: <http://groups.yahoo.com/group/bartssandbox>. Free membership is required.
- Main MySQL Web site: www.mysql.com
- Starting point for MySQL documentation: www.mysql.com/documentation/index.html
- www.oreillynet.com/lpt/wlg/3946
- For all things EJBGen related: www.beust.com/cedric/ejbgen
- Detailed information about JMS transactions and redelivery options: www.javaworld.com/javaworld/jw-03-2002/jw-0315-jms_p.html
- DuBois, Paul. *The Definitive Guide to Using, Programming, and Administering MySQL 4 Databases*, Second Edition. (www.bookpool.com/.x/d4jha9om4m/sm/0735712123)
- Oracle's deferrable constraint support: (http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96524/c22integ.htm#4666). Free membership to Oracle Technical Network (OTN) may be required.
- Checkout J2EE Patterns here: (<http://java.sun.com/blueprints/patterns/index.html>). For Session Façade Pattern used in the sample application refer to <http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html> and <http://java.sun.com/blueprints/patterns/SessionFacade.html>.
- For Value Object Pattern used in the sample application refer to <http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html> and <http://java.sun.com/blueprints/patterns/TransferObject.html>.
- JUnit, including downloading the software: www.junit.org/index.html



TROUBLE SEEING BEA PORTAL DATA?

NEW! **BEA PORTAL REPORTING SOLUTIONS**

POWERED BY THE FORMULA ONE E-REPORT ENGINE

- 100% Pure Java:** API-driven toolsets for building queries and creating reports against BEA Portal Server events.
- Visual Access to BT_EVENT:** Point-and-click wizards streamline report creation against the complex structure of BEA Portal Event and Behavior Tracking Data.
- Deliver Reports Through BEA Portal Server:** Embed reports that inherit the server features (security, connection pooling, etc.) of your BEA platform. No external report server to set up or maintain.
- Analyze Server Events:** Determine user traffic patterns. Find the most and least popular items in your Portal applications. Calculate click-throughs on marketing campaigns. And more!
- Flexible:** Available as an integrated WebLogic Workshop Extension or as a set of JAR files for use with JBuilder, IntelliJ IDEA, and other Java development environments.

FORMULA ONE
Enterprise Reporting Toolsets for Java Projects & Portals

ReportingEngines
JAVA REPORTING TOOLS FROM ACTIVATE

888-884-8665 • sales@reportingengines.com • <http://www.reportingengines.com/products/beaportal/overview.jsp>

Copyright © 2004 ReportingEngines (a division of Actuate Corporation). All rights reserved. Formula One is a registered trademark of Actuate Corporation. Java and Java-based trademarks and logos are the trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries. All other trademarks are property of their respective owners. All specifications subject to change without notice.



JMX Debugging

THIRD-PARTY TOOLS MAKE ACCESS EASY

BEA WebLogic 8.1 implements the Java Management Extensions (JMX) 1.0. Most WebLogic subsystems (JMS Providers, the JDBC Container, ExecuteQueues, etc.) and their constituents are instrumented as MBeans and contain attributes by which they can be configured, monitored, and managed. An administrative server instance implements an MBeanServer through which its configuration and runtime MBeans and those of its managed servers may be accessed. A managed server instance implements an MBeanServer through which only the configuration and runtime beans that are resident on it may be accessed.

In a debugging scenario in which a running server is misbehaving, both its configuration beans can be examined to diagnose the problem. BEA Weblogic MBeans may be accessed programmatically via the Java API. In addition, WebLogic provides two mechanisms for accessing MBeans without programming: the `wlconfig` command and the `wlconfig` task. There are, as well, numerous third-party tools that provide convenient access to MBeans. Two, which I will use in what follows, are `wlsh`, developed by Paco Gomez of BEA, and `wlsScripting`, developed using JPython by Satya Ghattu, also of BEA.

I have `TestDomain` configured with one cluster of two managed servers and an additional independent managed server. Connecting to the admin server with `wlsh` produces the result in Listing 1.

Connecting to `ManagedOne` produces Listing 2. Now I can retrieve the `ExecuteThreadCurrentIdleCount` from the admin server with

```
wlsh TestDomain: />
idleCount=/ExecuteQueueRuntime/weblogic.kernel.Default/
ExecuteThreadCurrentIdleCount
```

```
variable idleCount set to /ExecuteQueueRuntime/weblog-
ic.kernel.Default/ExecuteThreadCurrentIdleCount
(java.lang.String)
wlsh TestDomain: /> get $idleCount
15
```

It is almost always important in a production scenario for a crashed server to be restarted immediately. Listing 3, using `wlsScripting`, periodically checks the server's status and restarts it if necessary. At the same time, it samples critical attributes for debug purposes.

Listing 4 shows the output of running the script against an instance of the `examplesServer` as a thread becomes stuck.

In a production environment where large amounts of debug are a problem, it is often difficult to track down transient errors. The JMX API can be used to report debug only in certain conditions. Listing 5 installs a counter monitor that is triggered when the `OpenSocketsCurrentCount` attribute of the `ServerRuntime` MBeans reaches a certain level. It doesn't do anything in the `NotificationListener.handleNotification` method currently, other than note that a notification was received; however, in a real scenario you might launch a thread at this point that would report periodically on the values of attributes associated with critical resources, say the `PendingRequestsCurrentCount` from the `ExecuteQueueRuntime-MBean` for the default queue.

This is the output of running the `DebugMonitor`:

```
Java debug.DebugMonitor

Active Domain: medrecTwo
Active Servers:
  Name: MedRecServer
  ListenAddress: JWEAVER1/10.62.3.106
  ListenPort: 7001

  Number of servers active in the domain: 1
>>> MyCounter got notification: javax.management.moni-
tor.MonitorNotification
...
```



BY JIM WEAVER

AUTHOR BIO...

Jim Weaver is a backline WebLogic support engineer in Denver. He has been with BEA for more than 3 years and worked previously as a C++/Java developer for more than 10 years.

CONTACT...

jweaver@bea.com

REPRODUCED WITH PERMISSION FROM BEA SYSTEMS.

Reach Over 100,000

Enterprise Development Managers & Decision Makers with...



Offering leading software, services, and hardware vendors an opportunity to speak to over 100,000 purchasing decision makers about their products, the enterprise IT marketplace, and emerging trends critical to developers, programmers, and IT management

Don't Miss Your Opportunity to Be a Part of the Next Issue!

Get Listed as a Top 20* Solutions Provider

For Advertising Details Call 201 802-3021 Today!

*ONLY 20 ADVERTISERS WILL BE DISPLAYED. FIRST COME, FIRST SERVED.



The World's Leading iTechnology Publisher

Links

- **Starting point for WebLogic JMX documentation:**
<http://edocs.bea.com/wls/docs81/jmx/index.html>
- **Programming examples:**
<http://edocs.bea.com/wls/docs81/jmx/index.html>
- **weblogic.Admin command:**
http://edocs.bea.com/wls/docs81/admin_ref/cli.html#MBean_Management_Command_Reference
- **wlconfig Ant task:**
http://edocs.bea.com/wls/docs81/admin_ref/ant_tasks.html#1023006
- **wlsh:** www.wlshell.net
- **wlsScripting tool:** <http://dev2dev.bea.com/codelibrary/code/wlst.jsp>

Listing 1

```
wlsh [not connected]> connect t3://localhost:7771 system weblogic
connecting to t3://localhost:7771 as system...done

connection information:
- the server name is "myserver" and belongs to "TestDomain" domain
- this server is running as the administration server
- the shell is accessing all MBeans in this domain
- the server version is:
WebLogic Server 8.1 SP2 Fri Dec 5 15:01:51 PST 2003 316284
WebLogic XMLX Module 8.1 SP2 Fri Dec 5 15:01:51 PST 2003 316284
- see /ServerRuntime/myserver for more runtime information.

connected to myserver*

Searching for MBeans in myserver...done
- Domains: [JMImplementation, Security, TestDomain, WeblogicManagement,
weblogic]
- Types: 95
- MBeans: 762

accessing jndi@myserver...done

wlsh TestDomain:/>
```

Listing 2

```
wlsh [not connected]> connect t3://localhost:7772 system weblogic
connecting to t3://localhost:7772 as system...done

connection information:
- the server name is "ManagedOne" and belongs to "TestDomain" domain
- this server is running as a managed server
- the shell is accessing the MBeans in this server only
- the server version is:
WebLogic Server 8.1 SP2 Fri Dec 5 15:01:51 PST 2003 316284
WebLogic XMLX Module 8.1 SP2 Fri Dec 5 15:01:51 PST 2003 316284
- see /ServerRuntime/ManagedOne for more runtime information.

connected to ManagedOne*

Searching for MBeans in ManagedOne...done
- Domains: [JMImplementation, Security, TestDomain, WeblogicManagement]
- Types: 44
- MBeans: 198

accessing jndi@ManagedOne...done

wlsh TestDomain:/>
```

Listing 3

```
# This script loops infinitely, checking the status of a given server
every 5 seconds and
# starts the server if that server state changes from RUNNING to any.
Adapted from startMS.py
# distributed with wlsScripting from
http://dev2dev.bea.com/codelibrary/code/wlst.jsp

import thread
import time

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

def checkHealth(serverName):
    while 1:

        status = getAttrVal( 'State',
getBean(serverName,'ServerLifecycleRuntime'))
        print 'Status of Server ' + serverName + ' is ' + status

        if status != "RUNNING":
            print 'Starting server ' + serverName
            start(serverName)
        else:
            print 'idle default threads: ' + \

getAttrVal('ExecuteThreadCurrentIdleCount',getBean('weblogic.kernel.Default
t','ExecuteQueueRuntime')).toString()
            print 'pending requests : ' + \
                getAttrVal('PendingRequestCurrentCount',getBean('weblogic.ker-
nel.Default','ExecuteQueueRuntime')).toString()
            print 'open sockets : ' + \

getAttrVal('OpenSocketsCurrentCount',getBean(serverName,'ServerRuntime')).
toString()

        st =
getAttrVal('StuckExecuteThreads',getBean('weblogic.kernel.Default','Execut
eQueueRuntime'));
        if st != None:
            print st.__len__().toString() + ' STUCK THREADS'
            time.sleep(5)
```

```
def getBean(name,type):
    beans = home.getMBeansByType(type, home.getDomainName()).itera-
tor()
    bean = None
    while beans.hasNext():
        bean = beans.next()
        if bean.getName() == name:
            return bean
    return eqrtBean

def getAttrVal(attr,bean):
    return mbs.getAttribute(bean.getObjectNames(),attr);

checkHealth(serverName)
```

Listing 4

```
D:\> java weblogic.WLST monitorMS.py

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands

Connecting to a BEA WebLogic Server instance running at t3://local-
host:7001 as username weblogic ...
```

Subscribe Today!

- INCLUDES -
FREE DIGITAL EDITION!
(The 100% usable version)
GET YOUR ACCESS CODE INSTANTLY!

FREE RESOURCE CD INCLUDED!

Information **STORAGE + SECURITY** Journal

www.ISSJournal.com

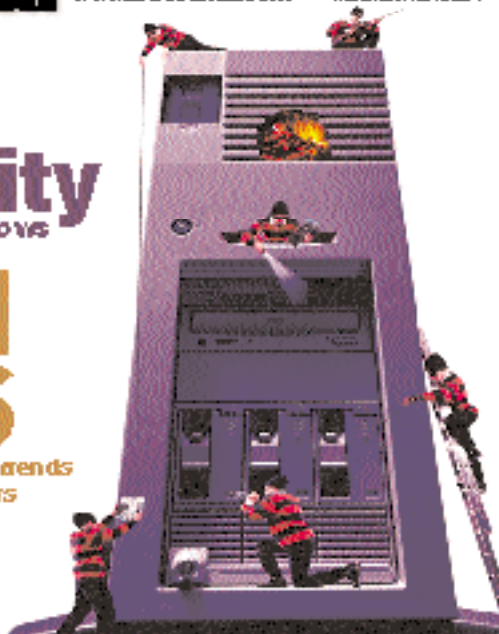
In this issue:

- IP Storage
- SAN NAS
- Disaster Recovery
- Enterprise Security
- Info security

Microsoft
RSA Security
SecureID for Windows

SAN NAS

Emerging technology trends and market maneuvers



The major info security issues of the day... identity theft, cyber-terrorism, encryption, perimeter defense, and more come to the forefront in ISSJ the storage and security magazine targeted at IT professionals, managers, and decision makers

Editorial Mission

Greater Collaboration and Efficiency Through Education

- ✓ ISSJ's editorial mission is to showcase proven solutions that will guide, motivate, and inspire senior IT and business management leaders in the planning, development, deployment, and management of successful enterprise-wide security and storage solutions.
- ✓ ISSJ brings together all key elements of data, storage and protection, and presents compelling insight into the benefits, efficiencies and effectiveness gained by focusing on these two critical areas of IT simultaneously.
- ✓ ISSJ is an objective, critical source of information that helps storage and security managers make informed management decisions about what is working today, and what they need to plan for tomorrow, and is the only publication that focuses exclusively on the needs of IT professionals who are driving the enterprise storage architecture/infrastructure while pursuing and incorporating the latest security technologies.
- ✓ ISSJ achieves our mission by delivering in-depth features, practical "how-to" solutions, authoritative commentary, hard-hitting product reviews, comprehensive real-world case studies, and successful business models, written by and for professional IT storage and security practitioners.

SAVE 50% OFF!

REGULAR NEWSSTAND PRICE

Only **\$39⁹⁹** ONE YEAR 12 ISSUES

www.ISSJournal.com

OR

1-888-303-5282

AMEDIA

The World's Leading IT Knowledge Publisher

Successfully connected to Admin Server 'examplesServer' that belongs to domain 'examples'.

```
Status of Server examplesServer is RUNNING
idle default threads: 15
pending requests : 0
open sockets : 1
Status of Server examplesServer is RUNNING
idle default threads: 15
pending requests : 0
open sockets : 1
...
idle default threads: 14
pending requests : 0
open sockets : 2
Status of Server examplesServer is RUNNING
idle default threads: 14
pending requests : 0
open sockets : 2
1 STUCK THREADS
```

Listing 5

```
package debug;

import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;
import java.util.HashMap;
import javax.management.*;
import javax.management.monitor.*;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.configuration.KernelMBean;
import weblogic.management.runtime.ServerStates;
import weblogic.management.*;

public class DebugMonitor {
    String url = "t3://localhost:7001";
    String username = "system";
    String password = "weblogic";
    String domainName = "";
    MBeanServer mbserver = null;

    public static void main(String[] args) {
        new DebugMonitor().init();
    }

    private void init() {
        MBeanHome home = null;

        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            domainName = home.getActiveDomain().getName();
            mbserver = home.getMBeanServer();
            System.out.println("Active Domain: " + domainName);

            System.out.println("Active Servers: ");
            Set mbeanSet = home.getMBeansByType("ServerRuntime");
            Iterator mbeanIterator = mbeanSet.iterator();
            String sn = ""; //server name
            ObjectName son = null; //server objname
```

```
        while(mbeanIterator.hasNext()) {
            ServerRuntimeMBean serverRuntime =
                (ServerRuntimeMBean)mbeanIterator.next();

            sn = serverRuntime.getName();
            son = serverRuntime.getObjectName();
            if(serverRuntime.getState().equals(ServerStates.RUN-
                NING)){
                System.out.println("\tName: " + sn);
                System.out.println("\tListenAddress: " +
                    serverRuntime.getListenAddress());
                System.out.println("\tListenPort: " +
                    serverRuntime.getListenPort());
            }
            System.out.println();
            System.out.println("\tNumber of servers active in the
                domain: " +
                    mbeanSet.size());

            CounterMonitorDebugger monitor = new
                CounterMonitorDebugger(sn,"MyCounter", new Integer(10), new Integer(3),
                    "ServerRuntime", "OpenSocketsCurrentCount");
        }
    } catch (Exception e) {
        System.out.println("Exception: " + e.toString());
    }
}

class CounterMonitorDebugger implements RemoteNotificationListener {
    String serverName;
    String name;
    CounterMonitorDebugger( String sn, String mn, Number ths, Number
        ofst, String parentMB, String obsvAttr) {
        this.serverName = sn;
        this.name = mn;
        CounterMonitor monitor = new CounterMonitor();
        try {
            WebLogicObjectName mon = new
                WebLogicObjectName(name, "CounterMonitor", domainName);
            try {
                mbserver.unregisterMBean(mon);
            } catch (Exception x) { }
            WebLogicObjectName pon = new
                WebLogicObjectName(serverName, parentMB, domainName);
            WebLogicObjectName oon = new
                WebLogicObjectName("default", parentMB, sn, pon);
            monitor.setThreshold(ths);
            monitor.setNotify(true);
            monitor.setOffset(ofst);
            monitor.setObservedObject(oon);
            monitor.setObservedAttribute(obsvAttr);
            monitor.addNotificationListener(this,null,null);
            mbserver.registerMBean((CounterMonitor) monitor, mon);
            monitor.start();
        } catch (Exception x) {
            System.out.println("CounterMonitorDebugger(" + name + ")
                exception: " + x.toString());
            x.printStackTrace(System.out);
        }
    }

    public void handleNotification(Notification notif, Object hand-
        back) {
        System.out.println(">>> " + name + " got notification: " +
            notif.getClass().getName());
    }
}
```

Intersperse Manages the Risk in SOA Deployments

Over the past 25 years, enterprise application systems have advanced tremendously, increasing functional scope, geographic distribution, degree of interconnectedness, and speed of information delivery. While this evolution yielded enormous productivity benefits and competitive advantages, it also caused IT complexity to explode, as we've moved through computing phases: from the mainframe, through client/server, into web apps, and finally on to SOA applications. In addition to this increased complexity, each of these phases was shorter than the last, allowing IT teams less room for error (see Figure 1).

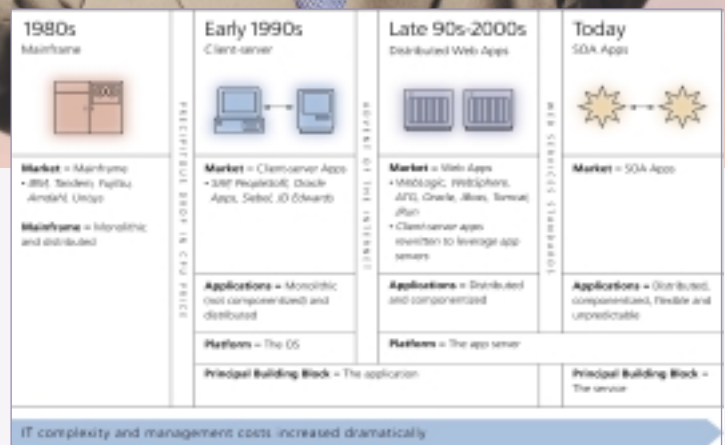
This all results in a relentless increase in risk – risk that has always been mitigated by IT management tools. New management solutions emerged each time we moved into a new phase, providing features and functions targeted to solve the specific needs of that phase.

This remains true today. Existing management tools were designed to address the needs of previous generations, and not to address the problems generated by today's advanced computing systems, which are increasingly built on J2EE and based on SOA principles.

Despite their many benefits, today's J2EE-based SOA application systems bring new challenges. Among other things, these systems:

- Are highly distributed, leveraging the Internet to coordinate business processes within and between enterprises.
- Have many more points of failure, as systems break down into granular software components, services, and interfaces, with dynamic, complex interdependencies, and relationships, any of which can cause failure either individually or through its interactions.
- Blur the line between application development and application integration, as systems are increasingly "built to integrate" with an emphasis on service reuse.

The BEA WebLogic Platform, from Workshop to WebLogic Integration, provides the market's most comprehensive toolset to design, build, and deploy SOA applications. The last piece of the production SOA puzzle is comprehensive, proactive management of SOA applications. This is a critical piece, as traditional management tools are insufficient to manage distributed, integrated, cross-application and cross-enterprise business processes – they were designed for another time and another world.



Intersperse Manager is the only product designed specifically to provide production management of SOA applications. It discovers, monitors, and manages J2EE-based SOA applications deployed on Application, Integration, Process, or Portal Servers, and it ensures the continuity of the vital, integrated business processes that run today's enterprises. By simultaneously managing both vertical applications and horizontal integrations, Manager delivers proactive production management of service-oriented BEA applications. Intersperse extends management to the complete BEA WebLogic Platform, including WebLogic Integration and WebLogic Portal.

Intersperse Manager is truly a production management solution, leveraging the increasingly ubiquitous Java Management Extensions (JMX) standard. JMX facilitates standardized discovery, monitoring, and management of components, applications, and services, as well as enabling hot deployment and removal from already running systems. This is in contrast to some tools today that have defaulted to the use of invasive, application-altering byte-code instrumentation as their means to gather management data.

To fully reap the benefits and mitigate the risks of J2EE-based SOA deployments, businesses need a new kind of management tool, one designed specifically to serve the needs of these environments. Intersperse Manager is the only management tool designed to help master the rapidly growing complexity of SOA business systems. It gives developers, operators, and analysts comprehensive visibility into all relevant tiers, tools to proactively monitor and analyze system performance in business contexts, and the control to automatically correct problems in production. As organizations evolve their business systems to BEA-based SOAs, they cannot afford to be without Intersperse Manager.

The BEA WebLogic

TRANSFER MESSAGES BETWEEN JMS PROVIDERS

What is a messaging bridge? And why and where would you use it?

A messaging system is one in which applications are loosely coupled through the exchange of messages. In crude terms it is like an e-mail system for applications.

A messaging bridge in turns moves messages between any two messaging systems/products. A messaging bridge consists of two destinations that are being bridged: a source destination from which messages are received, and a target destination to which messages are sent. Source and target bridge destinations can be either queues or topics.

BEA WebLogic introduced its messaging bridge in WebLogic 6.1, and has enhanced the features with later releases of WebLogic.

The WebLogic Messaging Bridge allows you to configure a forwarding mechanism between any two messaging products, thereby providing interoperability between separate implementations of BEA WebLogic JMS or between WebLogic JMS and another messaging product.

The WebLogic Messaging Bridge also provides the ability to specify a quality of service (QOS), as well as message filters, transaction semantics, and connection retry policies. Once a messaging bridge is configured, it is easily managed from the administration console, including temporarily suspending bridge traffic when necessary, tuning the execute thread pool size setting to suit your implementation, and monitoring the status of all your configured bridges.

Sample Application

This is a hypothetical application. It consists of two component applications, Stock Broker and Stock Exchange, that are deployed on Stock Broker Server and Stock Exchange Server respectively. Each component application should run independently of the other component, as well as communicating asynchronously with the other component. There is also a requirement that when one of the component applications sends a message the other component application should

receive it as as soon as its server is in a running state.

Stock Broker Server is an online server that handles user requests. It will be able to take a stock purchase request and display whether the purchase request succeeded or not, including the complete transaction history, if required. This server will not try to process the stock purchase request, but instead will send it to Stock Exchange Server for fulfillment.

Stock Exchange Server is an offline server that will receive messages from Stock Broker Server. It will receive the stock purchase requests and try to process them. This server might communicate with other external systems in order to fulfill the purchase request. For each stock purchase request received, this server will create a stock purchase response and will send the response back to Stock Broker Server.

In this implementation, Stock Broker is running on BEA WebLogic 8.1 and Stock Exchange is running on BEA WebLogic 7.0. WebLogic's JMS has been used for messaging on both servers, and WebLogic's Messaging Bridge has been used as a bridge between the two messaging systems. The choice of WebLogic 8.1 and WebLogic 7.0 along with WebLogic JMS messaging systems was here a matter of convenience. Conceptually, the messaging bridge can be used between any two messaging products.

As shown in Figure 1, Stock Broker Server has two queues, RequestSubmitQueue and ResponseReceiveQueue. Similarly, Stock Exchange Server has two queues, RequestReceiveQueue and ResponseSubmitQueue.

Stock Broker Server receives a stock purchase request from the user. SenderBean places the PurchaseRequestMessage on RequestSubmitQueue. Messaging Bridge1 forwards the message placed on RequestSubmitQueue of Stock Broker Server to RequestReceiveQueue of Stock Exchange Server.

Once a message is received on RequestReceiveQueue of Stock Exchange Server, ReceiveRequestMDB's (message driven bean) onMessage will be invoked by the container/server. ReceiveRequestMDB will get the object message and invoke ProcessRequestBean for fulfillment logic.



BY VIJAY CHINTA

AUTHOR BIO

Vijay Chinta is a senior software engineer at Nokia. His primary area of expertise is J2EE development for various enterprise applications in the telecommunications and transportation industries.

CONTACT...

vijay.chinta@nokia.com

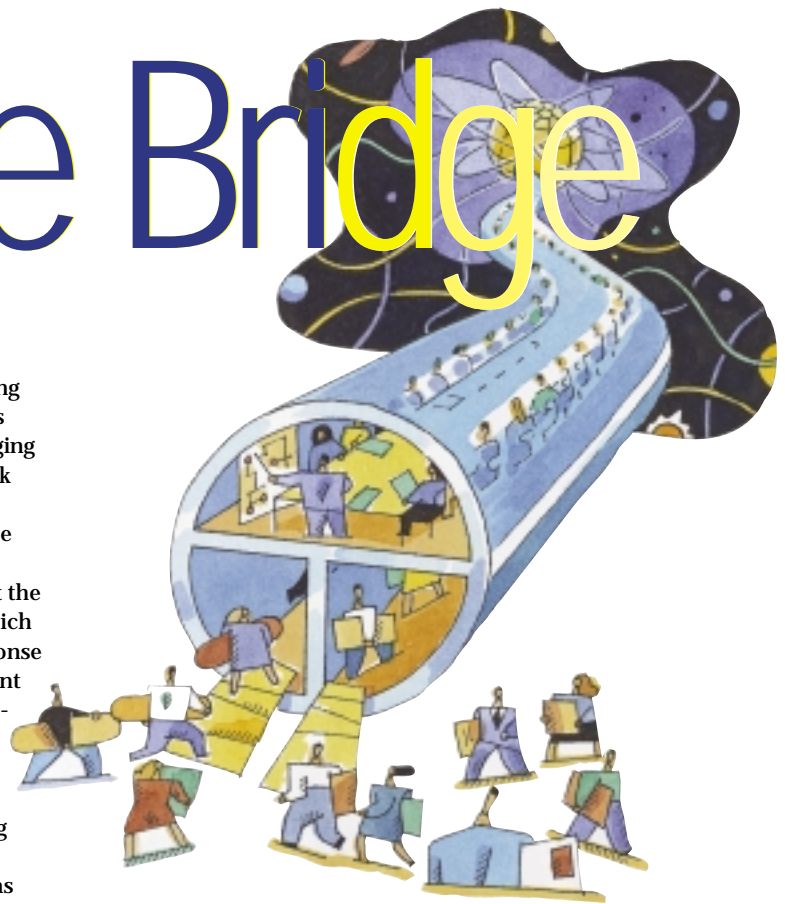
Message Bridge

ProcessRequestBean creates a PurchaseResponseMessage and invokes the SenderBean. The SenderBean is responsible for placing the PurchaseResponseMessage on the ResponseSubmitQueue. As soon as the message is placed on ResponseSubmitQueue, Messaging Bridge2 forwards that message to ResponseReceiveQueue of Stock Broker Server.

Once a message is received on ResponseReceiveQueue of the Stock Broker Server, ReceiveResponseMDBs onMessage will be invoked by the container/server. ReceiveResponseMDB will get the object message and invoke ProcessPurchaseResponseBean, which is responsible for processing the response and storing the response message to a flat file. The user can check the status of the current purchase request or the complete transaction history by accessing StockPurchaseResponse.jsp, which will invoke GetResponseFromStorage to get the list of response messages (transaction history) to be displayed.

In this sample application, any one or both of the messaging bridges can be turned off. At that time all the messages will be queued and will be forwarded to the target bridge destination as soon as the bridge is turned on again.

Similarly, if the server that is acting as a target destination of a messaging bridge is down (not in a running state), all of the messages will be queued and sent to the destination server's corresponding queue as soon as the target server becomes available (running state) again.



- **Stock Broker Server is brought up.** As soon as it is in a running state, Messaging Bridge2 forwards the response message from ResponseSubmitQueue to ResponseReceiveQueue. The Stock Broker Server stores the response in a flat file and displays it whenever the user wants to view it.

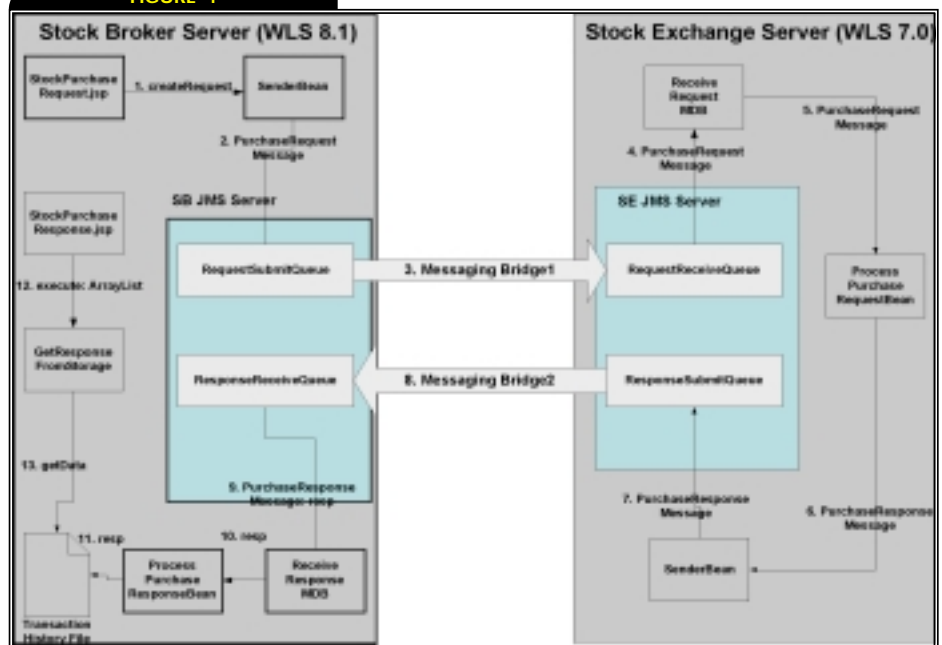
Example Scenario Demonstrating the Use of the Messaging Bridge

Let's assume that Stock Broker Server is in a running state, and Stock Exchange Server is brought down for regular maintenance or for some unforeseen reason.

- **A user enters a stock purchase** request on Stock Broker Server. The Stock Broker Server takes the user's request, places the request on RequestSubmitQueue.
- **Stock Exchange Server is brought up to a running state.** Messaging Bridge1 forwards the queued request to Stock Exchange Server. Stock Exchange Server processes/fulfills the request, creates a response message and places it in the ResponseSubmitQueue.

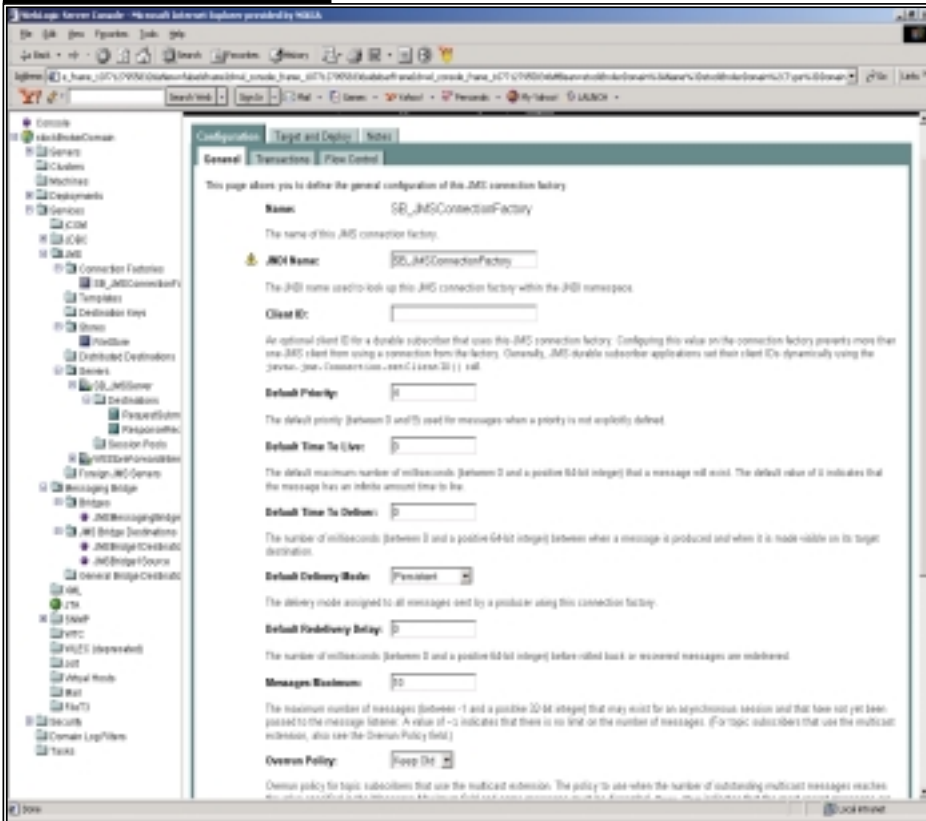
Now assume that Stock Broker Server becomes unstable and has to be brought down.

FIGURE 1



Stock purchase request

FIGURE 2



Values for connection factory

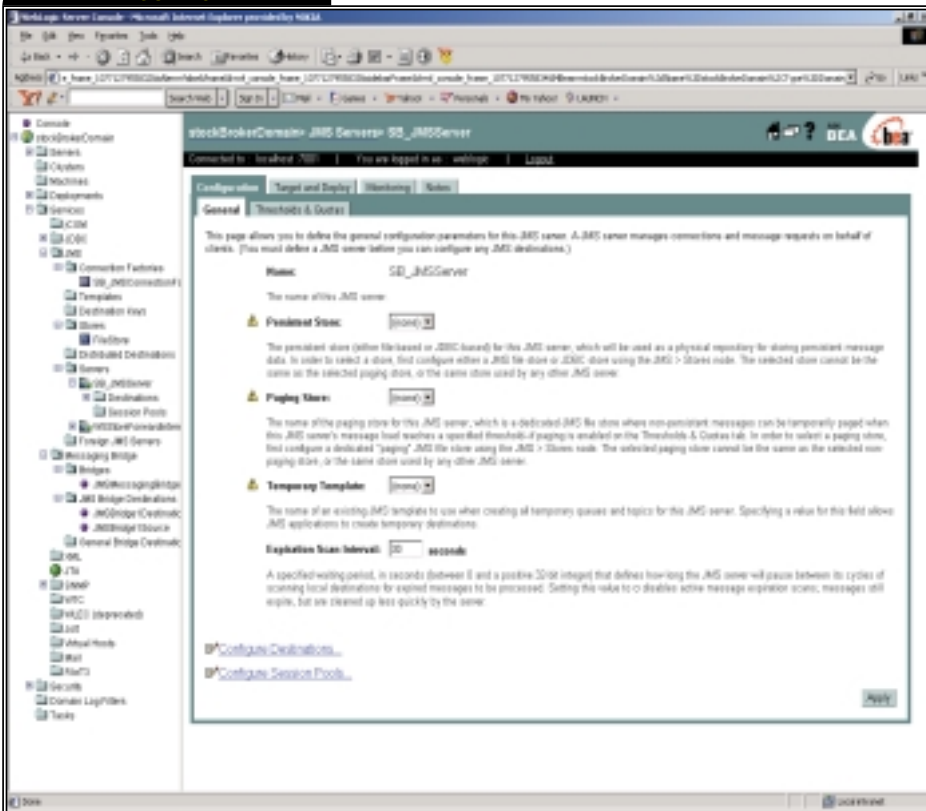
This is just one example, but as we saw above, no messages were lost and there was no application logic required to do any additional processing when the target server was down.

Features Provided by the Message Bridge

By using the message bridge in the sample application, there was no runtime dependency between the two applications. Any one of the applications can be down and it will not impact the second application. Basically, all the messages on the first server will be queued and the messaging bridge will automatically start sending messages to the target destination as soon as the second server is up and running. It provides flexibility when you have the Stock Broker Application and Stock Exchange Application deployed on different versions of WebLogic. The Stock Broker Application and Stock Exchange Application can run on any version of WebLogic, which allows loose coupling in terms of software infrastructure.

One more advantage is that it also provides future flexibility to go with a completely different JMS implementation (e.g., MQSeries) or a non-JMS messaging product (Tuxedo) for Stock Exchange Application.

FIGURE 3



Values for Stock Broker Server

Configuring JMX Queues

Following are the steps required in setting up the JMS Bridge. The steps required for both servers are provided. (*Note:* We will use the notation of step A to signify that we are working on Stock Broker Server [BEA WebLogicServer 8.1], whereas step B signifies that we are working on Stock Exchange Server [BEA WebLogic Server 7.0]. Also, the screenshots of the BEA WebLogic Administration Console are provided only for Stock Broker Server, as a very similar configuration was used on both servers.)

Configure a JMS Connection Factory FOR STOCK BROKER SERVER

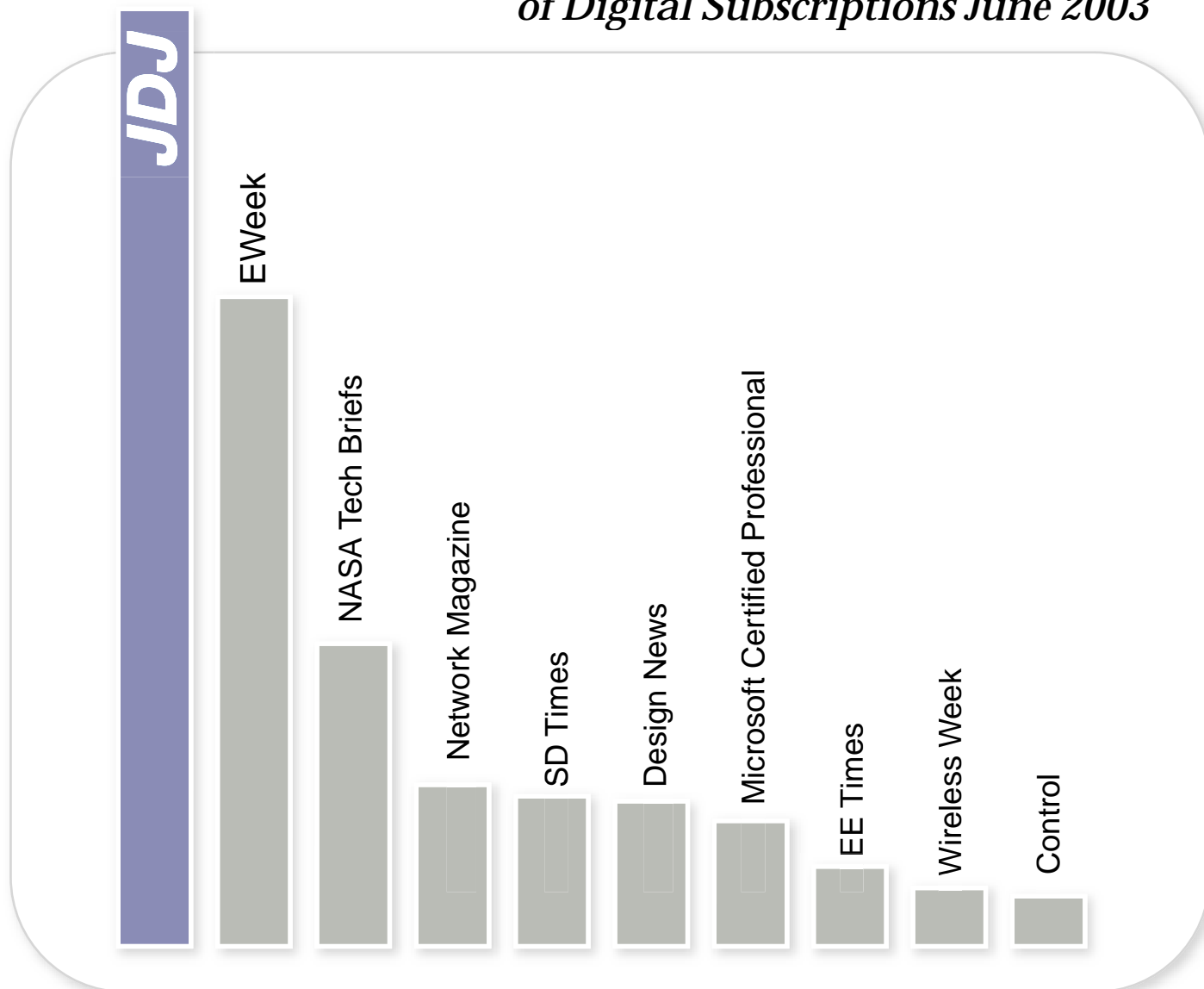
Using the WebLogic Administration Console, go to stockBrokerDomain > Services > JMS > Connection Factories. Click on "Configure a new JMS Connection Factory"

1. The Name and JNDI Name have been specified as: SB_JMSConnectionFactory.
2. All the other values have been set to default values (see Figure 2).
3. Select the Transactions tab and check the box for "XA Connection Factory Enabled". This is required if your application uses transactions across the mes-

Oops, we did it again!

JDJ No. 1 in the World

Top 10 Publications Ranked by Volume of Digital Subscriptions June 2003



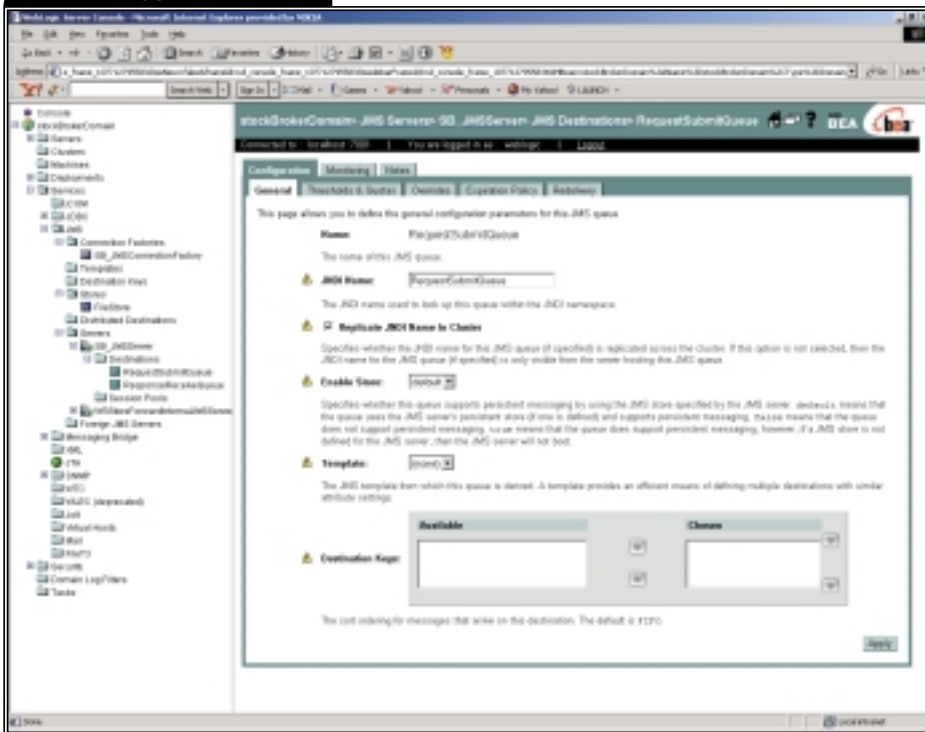
Source: BPA International publisher's statements
Note: Under BPA International rules, a subscriber who has opted to receive both the print and digital editions of a publication is reported only once and counted only once within that and the corresponding % of total qualified circulation of those combined subs are provided only for informational purposes.

Carmen Gonzalez
Senior VP Marketing



Miles Silverman
VP Marketing

FIGURE 4



Values for JMS queues on Stock Broker Server

saging bridge and if you specify the “Quality of Service” of the messaging bridge as “Exactly-once”.

FOR STOCK EXCHANGE SERVER

Using the WebLogic Administration Console, go to `stockExchangeDomain > Services > JMS > Connection Factories`. Click on “Configure a new JMS Connection Factory”.

1. The Name and JNDI Name have been specified as `SE_JMSConnectionFactory`.
2. All other values have been set to default values.
3. Select the Transactions tab, check the box for “User Transactions Enabled”, and click on the “Apply” button. Then check the box for “XA Connection Factory Enabled” and click on the “Apply” button. (“User Transactions Enabled” must be checked first, as there is a known bug in BEA WebLogic 7.0.) These steps are required if your application uses transactions across the messaging bridge and if you specify the “Quality of Service” of the messaging bridge as “Exactly-once”.

Configure a JMS Server

FOR STOCK BROKER SERVER

Using the WebLogic Administration Console, go to `stockBrokerDomain > Services > JMS > Servers`. Click on

“Configure a new JMS Server”.

1. The Name has been specified as `SB_JMSQueue`.
2. All other values have been set to default values (see Figure 3).

FOR STOCK EXCHANGE SERVER

Using WebLogic Administration Console, go to `stockExchangeDomain > Services > JMS > Servers`. Click on “Configure a new JMS Server”.

1. The Name has been specified as `SE_JMSQueue`.
2. All other values have been set to default values.

Configure JMS Queues

FOR STOCK BROKER SERVER

Using the WebLogic Administration Console, go to `stockBrokerDomain > Services > JMS > Servers > JMSServer > Destinations`.

1. Click on “Configure a new JMS Queue”.
 - a. The Name and JNDI Name have been specified as `RequestSubmitQueue`.
 - b. All the other values have been set to default values (see Figure 4).
2. Click on “Configure a new JMS Queue”.
 - a. The Name and JNDI Name have been specified as `ResponseReceiveQueue`.
 - b. All the other values have been set to default values.

FOR STOCK EXCHANGE SERVER

Using the WebLogic Administration Console, go to `stockExchangeDomain > Services > JMS > Servers > JMSServer > Destinations`.

1. Click on “Configure a new JMS Queue”.
 - a. The Name and JNDI Name have been specified as: `RequestReceiveQueue`.
 - b. All other values have been set to default values.
2. Click on “Configure a new JMS Queue”.
 - a. The Name and JNDI Name have been specified as `ResponseSubmitQueue`.
 - b. All the other values have been set to default values.

Deploy the JMS Transaction Adapter in the Application Domain

FOR STOCK BROKER SERVER

Using the WebLogic Administration Console, go to `stockBrokerDomain > Deployments > Connector Module`. Click on “Deploy a new Connector Module” and upload the `jms-xa-adp.rar` file from WebLogic’s `server/lib` directory.

FOR STOCK EXCHANGE SERVER

Using the WebLogic Administration Console, go to `stockExchangeDomain > Deployments > Connector Modules`. Click on “Deploy a new Connector Module” and upload the `jms-xa-adp.rar` file from the `WebLogic Server/lib` directory.

Configure JMS Bridge Destinations

FOR STOCK BROKER SERVER

Using the WebLogic Administration Console, go to `stockBrokerDomain > Services > Messaging Bridge > JMS Bridge Destinations`.

1. Click on “Configure a new JMS Bridge Destination”. These values have been set:

Name: `JMSBridge1Source`

Adapter JNDI Name: `eis.jms.WLSConnectionFactoryJNDIXA` (default value, but this resource adapter needs to be deployed for the bridge to function properly)

Connection URL: `t3://localhost:7001` (the URL of the Stock Broker Server)

Connection Factory JNDI Name: `SB_JMSConnectionFactory` (the connection factory that we created earlier)

Destination JNDI Name:

`RequestSubmitQueue` (the queue that `JMSBridge1Source` is trying to read the messages from)

Destination Type: `Queue` (optional username/password that the adapter will use to access this bridge destination). If you spec-

if username/password, the following step is required:

- a. Go to stockBrokerDomain > Security. Select Advanced tab
 - Uncheck "Enable Generated Credential".
 - Specify the new credential and make sure that it is set to the same value across the two application domains (stockBrokerDomain and stockExchangeDomain).

All other values have been set to default values (see Figure 5).

2. Click on "Configure a new JMS Bridge Destination". These values have been set:

Name: JMSBridge1Destination

Adapter JNDI Name:

eis.jms.WLSCONNECTIONFACTORYJNDIXA

(default value, but this resource adapter must be deployed for the bridge to function properly)

Connection URL: t3://localhost:7003 (the URL of the Stock Exchange Server, which is the server on which the bridge's target destination is running.)

Connection Factory JNDI Name:

SE_JMSCONNECTIONFACTORY (the connection factory that we created earlier).

Destination JNDI Name:

RequestReceiveQueue (the queue where JMSBridgeDestination will place the new messages that were transferred over the bridge)

Destination Type: Queue. (An optional username/password that the adapter will use to access this bridge destination can also be set. If you specify the username/password, please set up credentials.)

All other values have been set to default values.

CONFIGURE JMS BRIDGE DESTINATIONS FOR SERVER B

Using the WebLogic Administration Console, go to server2Domain > Services > Messaging Bridge > JMS Bridge Destinations.

1. Click on "Configure a new JMS Bridge Destination". These values have been set:

Name: JMSBridge2Source

Connection URL: t3://localhost:7003 (the URL of the Stock Exchange Server).

Connection Factory JNDI Name:

SE_JMSCONNECTIONFACTORY (the connection factory we created earlier).

Destination JNDI Name:

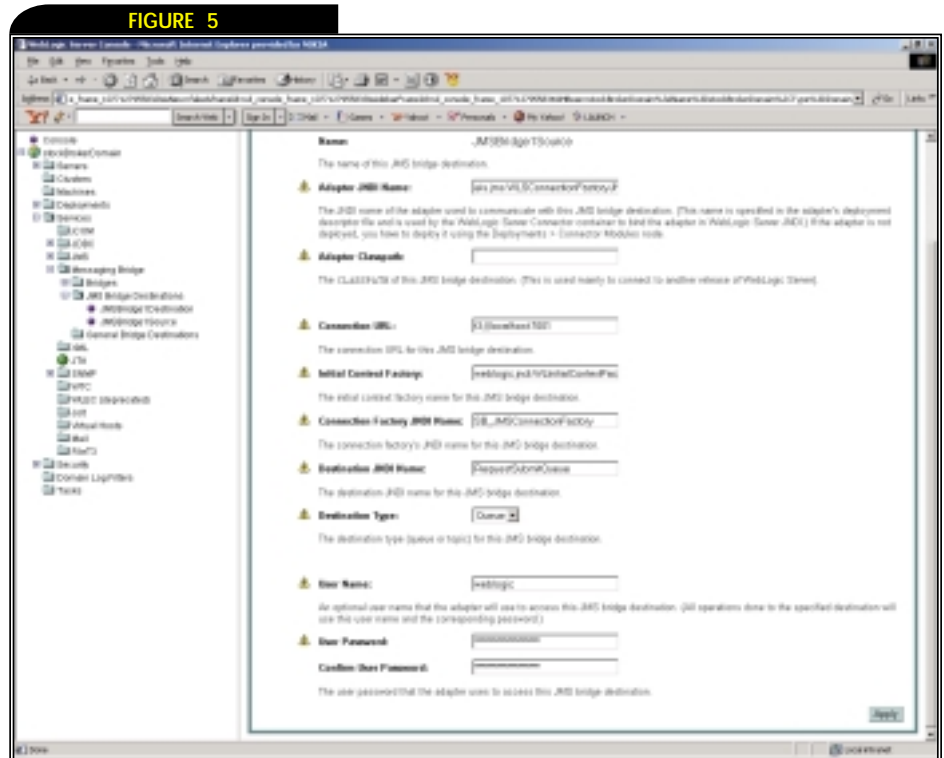
ResponseSubmitQueue (the queue that

JMSBridgeSource is trying to read the messages from)

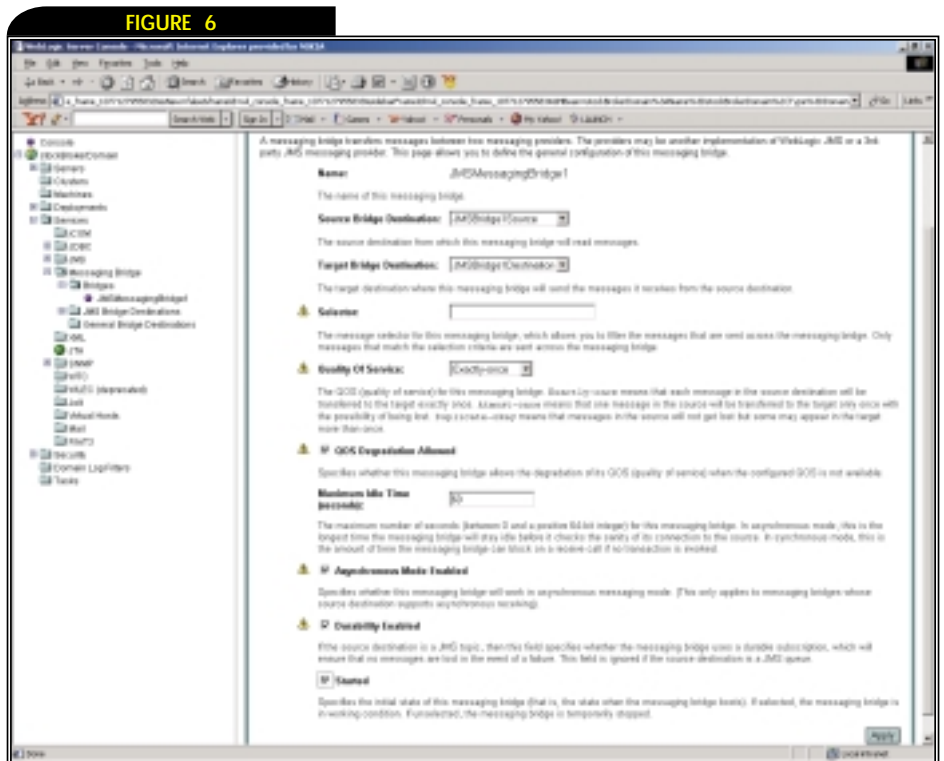
Destination Type: Queue. (An optional username/password that the adapter will use to access this bridge destination can also be

set. If you specify the username/password please set up credentials.)

All other values have been set to default values.



Values for JMS Bridge Destinations



Values for the messaging bridge

SUBSCRIBE TO THE FINEST TECHNICAL JOURNALS IN THE WORLD...

IT'S JUST A CLICK AWAY!

2. Click on “Configure a new JMS Bridge Destination”. The values have been set.

Name: JMSBridge2Destination
Connection URL: t3://localhost:7001 (the URL of the Stock Broker Server, which is the server on which the bridge's target destination is running).
connection factory JNDI Name: SB_JMSConnectionFactory (the connection factory we created earlier)
Destination JNDI Name: ResponseReceiveQueue (This is the queue in which JMSBridgeDestination will place the new messages that were transferred over the bridge)
Destination Type: Queue. (An optional username/password that the adapter will use to access this bridge destination can also be set. If you specify the username/password, please set up credentials.)

All other values have been set to default values.

Configure Messaging Bridge(s)
 ON STOCK BROKER SERVER
 (STOCK BROKER > STOCK EXCHANGE)

Using the WebLogic Administration Console, go to stockBrokerDomain > Services > Messaging Bridge > Bridges.

Click on “Configure a new Messaging Bridge Destination”. These values have been set:

Name: JMSMessagingBridge1
Source Bridge Destination: JMSBridge1Source
Target Bridge Destination: JMSBridgeDestination
Quality of Service: Exactly-once (Make sure that XA Connection Factory is enabled for the JMSConnectionFactory created earlier, along with deploying the bridge resource adapter described earlier.)

All other values have been set to default values (see Figure 6).

ON STOCK EXCHANGE SERVER
 (STOCK EXCHANGE > STOCK BROKER)

Using WebLogic Administration Console, go to stockExchangeDomain > Services > Messaging Bridge > Bridges. Click on “Configure a new Messaging Bridge Destination”. These values have been set:

Name: JMSMessagingBridge2
Source Bridge Destination: JMSBridge2Source (4.5.B.1)
Target Bridge Destination: JMSBridge2Destination (4.5.B.2)

Quality of Service: Exactly-once (make sure that XA Connection Factory is enabled for the JMSConnectionFactory created earlier, along with deploying the bridge resource adapter)

All other values have been set to default values.

Summary

The BEA WebLogic Messaging Bridge is a J2EE device provided by WebLogic (from version 6.1 onward) and is used to transfer messages between two JMS providers. You can use the message bridge to move messages from a destination in one JMS provider to destination in another JMS provider. (One example could be transferring a message in a queue on the WebLogic JMS server to a topic in Sonic MQ JMS Server)

In addition to offering interoperability between different JMS providers, the bridge

“The BEA WebLogic Messaging Bridge is a J2EE device provided by WebLogic [from version 6.1 onwards]”

also allows interoperability between different versions of WebLogic JMS. Although the bridge itself will run on version 7.0 or greater, it can be used to forward messages to destinations running on version 5.1 and greater.

No coding is required; it is purely configuration. There is a built-in capability to ensure quality of service and connection management (Exactly-once, Duplicates-okay, and Atmost-once). It is easy to configure multiple bridges, and there is an ability to dynamically start and stop individual bridges.

References

- *Configuration of Messaging Bridge:* http://edocs.bea.com/wls/docs81/ConsoleHelp/domain_messagingbridge_config_general.html
- *Console Help:* http://edocs.bea.com/wls/docs70/ConsoleHelp/messaging_bridge.html
- *Using the BEA WebLogic Messaging Bridge:* <http://edocs.bea.com/wls/docs70/adminguide/msgbridge.html>
- *Enterprise Integration Patterns:* www.enterpriseintegrationpatterns.com/MessagingBridge.html



Create software so brilliant
it can manage itself.

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>





MANAGEMENT

Application Management with WebLogic Server for Developers PART 6

CUSTOM APPLICATION MANAGEMENT USING JMX



BY ROBERT PATRICK & VADIM ROSENBERG

AUTHOR BIOS...

Robert Patrick is a director of technology in BEA's CTO Office and coauthor of the book *Mastering BEA WebLogic Server: Best Practices for Building and Deploying J2EE Applications*. Robert has spent his career helping customers design, build, and deploy high-performance, fault-tolerant, mission-critical distributed systems using BEA Tuxedo and BEA WebLogic Server.

Vadim Rosenberg is the product marketing manager for BEA WebLogic Server. Before joining BEA two years ago, Vadim had spent 13 years in business software engineering, most recently at Compaq Computers (Tandem Division) developing a fault-tolerant and highly scalable J2EE framework.

CONTACT...

rpatrick@bea.com
vadimr@bea.com

REPRODUCED WITH PERMISSION FROM BEA SYSTEMS.

This article is the last in a series on BEA WebLogic Server administration and management for developers.

The first installment focused on the WebLogic Server administration concepts and terminology, and the graphical tools for packaging an application and setting up and configuring a WebLogic Server domain. The second article looked at the available application deployment, run time management, and the monitoring facilities that did not require knowledge of JMX. The third article discussed the basic concepts and terminology of JMX and the WebLogic Server 8.1 JMX infrastructure, and showed you how to use JMX-specific tools that come with WebLogic Server 8.1. The fourth article focused on the basics of how to write custom Java applications that use JMX to configure, administer, and manage WebLogic Server 8.1-based applications. Last month, we continued our discussion of JMX programming by showing you how to use the notification facilities to create notification listeners, monitors, and timers (see **WLDJ**, Vol. 2, issues 10–12; Vol. 3, issues 2 and 4).

This month we'll describe how to expose your own management functionality through JMX and ultimately through the Weblogic Admin Console. We start off by discussing why you might choose

to instrument your application with JMX. Then, we discuss the basics of instrumenting your application using standard, dynamic, and model MBeans. Finally, we show you how to expose your MBeans via the WebLogic Console.

Choosing to Instrument with JMX

Why would you want to consider instrumenting your application using JMX?

If you read the previous articles in this series, you should already know the answer to this question. If not, the answer is simple: application code that is instrumented using JMX allows the application's configuration and runtime information to be monitored and managed just like any other resource in WebLogic Server.

Last month, we showed you how JMX notification and monitors allow your application to dynamically monitor and track the values of any number of MBeans. Exposing the application information through an MBean makes it easy to correlate your application state with that of other MBeans, such as the WebLogic MBeans that surface statistics about the various servers and application components that make up your production environment.

Finally, instrumenting your code with JMX allows you to expose the monitoring and management functionality of your application through the WebLogic Admin Console.

What sort of functionality might you expose through JMX? The answer is any attribute or

SYS-CON MEDIA

304,187 of the World's Foremost IT Professionals
DIRECT MAIL, EMAIL OR MULTI-CHANNEL

Target CTOs, CIOs and CXO-level IT professionals and developers who subscribe to SYS-CON Media's industry leading publications

Java Developer's Journal...
 The leading publication aimed specifically at corporate and independent java development professionals



PowerBuilder Developer's Journal...
 The only PowerBuilder resource for corporate and independent enterprise client/server and web developers



ColdFusion Developer's Journal...
 The only publication dedicated to ColdFusion web development

LinuxWorld Magazine...
 The premier monthly resource of Linux news for executives with key buying influences



WLDJ...
 The official magazine for BEA WebLogic application server software developers, IT management & users

Web Services Journal...
 The only Web Services magazine for CIOs, tech, marketing & product managers, VARs/ISVs, enterprise/app architects & developers



.NET Developer's Journal...
 The most read iTechnology publication for Windows developers & CXO management professionals

XML-Journal...
 The world's #1 leading XML resource for CEOs, CTOs, technology solution architects, product managers, programmers and developers



WebSphere Developer's Journal...
 The premier publication for those who design, build, customize, deploy, or administer IBM's WebSphere suite of software



Wireless Business & Technology...
 The wireless magazine for key corporate & engineering managers, and other executives who purchase communications products/services

Recommended for a variety of offers including Java, Internet, enterprise computing, e-business applications, training, software, hardware, data back up and storage, business applications, subscriptions, financial services, high ticket gifts and much more.

NOW AVAILABLE!
 The SYS-CON
 Media Database
 304,187 postal
 addresses



For email information:
 contact Frank at 845-731-3832
 frank.cipolla@epostdirect.com
 epostdirect.com 800-409-4443 fax 845-620-9035

For postal information:
 contact Kevin at 845-731-2684
 kevin.collaply@edithroman.com
 edithroman.com 800-223-2194 fax 845-620-9035





functionality of your application that you want to be able to manage. For example, an application might use a logging framework that allows logging levels to be changed dynamically at runtime. By instrumenting the logging framework to expose the logging level through JMX, it is now relatively easy to:

- Change the logging level dynamically through the normal JMX mechanisms.
- Use JMX notification and monitors to change the logging level whenever certain other conditions on other MBeans are observed.
- Expose the logging framework configuration and/or runtime information through the WebLogic Console.

Now that you know what functionality you want to expose, the next decision that you need to make is what type of MBean you want to implement. JMX provides a number of alternatives for implementing MBeans:

- Standard MBeans
- Dynamic MBeans
- Model MBeans

We will use the same simple example to illustrate these three approaches side by side (the examples are online at www.sys-con.com/weblogic/sourcecode.cfm). Our examples use a simple Java class that simulates a logging framework and exposes the current logging level and the number of log records written through an MBean. We include a simple Web application that allows you to write messages to the log (which, in our example, is stdout). These examples demonstrate a simple case where the developer wants to present state and usage information about some portion of their application in a way that the system administrator or operations staff can easily monitor it. Later, we will show you how to expose this functionality through the WebLogic Admin Console.

JMX Instrumentation Using Standard MBeans

Implementing a standard MBean is about as simple as it gets. Before you start, the MBean interface class and MBean implementation class names must follow the specified pattern for all standard MBeans: the MBean interface class name must be the same as the MBean implementation class name with the suffix MBean appended. In our example, the MBean implementation class name is `wldj.standard.Logger`. Therefore, the MBean interface class name is `wldj.standard.LoggerMBean`. Now you're ready to create your standard MBean.

First, create an interface class that exposes the JMX attributes and operations through its method definitions. As we discussed in part 3, the attributes are exposed through getter and/or setter methods that conform to the standard JavaBean formats:

```
public Bar getFoo();
public void setFoo(Bar newValue);
```

All other methods exposed in the interface that do not follow these conventions are considered JMX operations. Our `LoggerMBean` interface exposes the following JMX attributes and operations:

Read-Write Attributes:

`LoggingLevel`

Read-Only Attributes:

`DebugLogWrites`

`InfoLogWrites`

`WarningLogWrites`

`ErrorLogWrites`

`EmergencyLogWrites`

`TotalLogWrites`

Operations:

`void resetLogWriteStatistics()`

Next, create your MBean implementation class to implement the MBean interface you defined. Remember, your interface only needs to expose the methods that you want JMX applications to be able to manage.

Finally, you need to include initialization code in your application to create and register your MBean with the `MBeanServer`. In our example, we used a `WebLogic Startup` class to create and register the MBean with the MBean server, as shown in the code snippet from the `LoggerStartup` class (see Listing 1).

The primary limitation of this approach is that it does not allow you to expose additional descriptive information about the attributes. JMX 1.2 addresses this limitation by providing a `standardMBean` base class, which you can extend and use to provide that additional information. You will, however, have to wait until the next release of BEA WebLogic Server to take advantage of this feature.

JMX Instrumentation Using Dynamic MBeans

Implementing a Dynamic MBean is a little more complex but gives you much more flexibility in that you can actually expose JMX attributes and operations that are defined at runtime rather than at compile time. To implement a Dynamic MBean, your MBean implementation class must implement the `DynamicMBean` interface, which includes:

- Providing an `MBeanInfo` object that details information about the JMX attributes, operations and notifications that are provided by the MBean
- Implementing the `getAttribute()` and `setAttribute()` methods that provide read and write capabilities for the MBean's JMX attributes
- Implementing the `invoke()` method that interprets the method and attribute types and performs the desired JMX operation
- Implementing the `NotificationBroadcaster` interface if the MBean emits JMX notifications

SAVE 17% OFF

DEVELOPER'S PowerBuilder Journal

OFFER SUBJECT TO CHANGE WITHOUT NOTICE **12 Issues for \$149⁹⁹**

• New PowerBuilder features • Tips, tricks, and techniques in server-side programming • DB programming techniques • Tips on creating live PowerBuilder sites • Product reviews • Display ads for the best add-on products

That's a savings of **\$31** off the annual newsstand rate. Visit our site at www.sys-con.com/pbdj/ or call **1-800-303-5282** and subscribe today!

- Implementing the MBeanRegistration interface if the MBean needs lifecycle callbacks to create and/or cleanup MBean state.

In our example, we implement the DynamicMBean interface methods. The implementations are fairly straightforward so we will not go through the details here (see the example code).

Dynamic MBeans provide the greatest flexibility and allow the use of the entire JMX functionality. As the example demonstrates, it is also a great deal more effort than implementing a standard MBean. How do you retain the flexibility of dynamic MBeans while reducing the complexity? Model MBeans significantly reduce the level of effort needed to implement dynamic MBeans.

JMX Instrumentation Using Model MBeans

Model MBeans provide similar capabilities to dynamic MBeans but are significantly easier to implement. The majority of the effort in using model MBeans is in constructing the metadata to describe the attributes, constructors, operations, and notifications that the MBean will expose and how those relate to the application class being instrumented. Many of the benefits of model MBeans come from their ability to adapt preexisting implementation code without modification.

We chose to put the logic for creating the ModelMBeanInfo object that contains the metadata inside the Logger class itself. We could have just as easily put this in the LoggerStartup class so that the Logger class would have no JMX code whatsoever. In our example, we used the version of the ModelMBeanAttributeInfo class constructor that assumes the JMX attribute being exposed has the standard getter and setter method names on the instrumented class.

Model MBeans also support changing the names of the attributes and operations by using the Descriptor class to specify different method names on the instrumented class. The descriptor contains a list of name-value pairs. The JMX 1.0 specification defines a set of required and optional descriptor fields. For example, setting the currencyTimeLimit field in the descriptor causes the model MBean implementation class to cache the value of the attribute for the specified period of time. This flexibility is important when acquiring the value

involves making a remote invocation or querying a back end system. Since our class was developed from a standard MBean, there was no need to use the descriptor functionality.

Once the ModelMBeanInfo object is populated, we simply need to create an instance of a ModelMBean using the RequiredModelMBean class, configure it to instrument our Logger class, and register it with the MbeanServer (see Listing 2).

While the code for creating and populating the ModelMBeanInfo object is more work than creating a standard MBean, it is equivalent to implementing the DynamicMBean.getMBeanInfo() method. For this effort, you get the flexibility of a dynamic MBean without the need to implement the other dispatch methods in the DynamicMBean interface.

Surfacing the MBean in the WebLogic Admin Console

The WebLogic Admin Console provides an extensibility mechanism that allows you to add pages to the Console to expose your application's MBeans. The basic steps are:

1. Implement a Java class that extends the weblogic.management.console.extensibility.Extension class and implements the weblogic.management.console.extensibility.NavTreeExtension interface.
2. Write a JSP that describes the nodes that you want to add to the Console's navigation tree.
3. Write a set of JSP pages that implement the pages that appear on the right hand side of the console whenever the user selects the nodes from the navigation tree.
4. Create a set of Web application deployment descriptors that tell the server that this web application is a console extension.
5. Package the Web application to the necessary files and deploy the Web application as you would any other Web application.

When writing the Java class, the most important method to implement is the getNavExtensionFor() method. The argument passed in is an MBean reference that indicates the different nodes in the navigation tree. Your application's response to these calls determines about where the new node will appear in the navigation tree. For example, we want our extension to show up at the domain level so our imple-

Once you're in it...



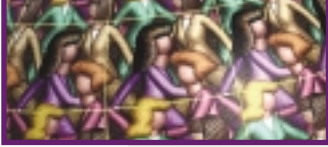
...reprint it!

- ColdFusion Developer's Journal
- Java Developer's Journal
- Web Services Journal
- Wireless Business & Technology
- MX Developer's Journal
- PowerBuilder Developer's Journal
- .NET Developer's Journal
- LinuxWorld Magazine
- WLDJ

Contact Kristin Kuhnle
201 802-3026
kristin@sys-con.com

REprints

SYS-CON
MEDIA



mentation only returns a URL if the argument is an instance of the DomainMBean:

```
public String getNavExtensionFor(Object key)
{
    if (key instanceof DomainMBean)
        return "logging_navlink.jsp";
    return null;
}
```

To describe the nodes you want to add, you must write the JSP file whose name matches the value returned from the `getNavExtensionFor()` method. BEA WebLogic Server 8.1 provides a custom JSP tag library, `console_extensions_taglib.tld`, that contains the JSP custom tag `wl:node` that you will need to use to define your Console extension nodes. To create a nested set of nodes, you simply nest the `wl:node` tags in the JSP. The `url` parameter to the `wl:node` tags defines which JSPs are called when the user selects one of your custom nodes in the console. For example, selecting the standard MBean Loggers folder in our example causes the `standard_summary.jsp` to be invoked to fill in the right hand side of the Console window (see Listing 3).

When writing the set of Console pages for your application, use the `wl:tag` JSP custom tag to create a set of tabs similar to the tabbed pages in the other portions of the WebLogic Console. This tag allows you to create up to two levels of tabs. Look at our example to see how to use this functionality. For more information about the custom

JSP tags provided to support extending the WebLogic Console, see the online documentation at http://edocs.bea.com/wls/docs81/console_ext/taglib.html.

Next, you need to write the Web application deployment descriptors. There are three things that you must do in addition to the normal information to describe the web application's contents. First, you need to tell the Console how to find your `NavTreeExtension` class by defining a `context-param` in the `web.xml` deployment descriptor:

```
<context-param>
  <param-
name>weblogic.console.extension.class</param-
name>
  <param-
value>wldj.LoggingConsoleExtension</param-value>
</context-param>
```

Second, you need to add the `taglib` definition to the `web.xml` deployment descriptor:

```
<taglib>
  <taglib-
uri>console_extension_taglib.jar</taglib-uri>
  <taglib-location>
    WEB-INF/console_extension_taglib.tld
  </taglib-location>
</taglib>
```


Finally, you need to set the `CookieName` that your Web application uses to be the same as the one used by the WebLogic

Admin Console so that your extension will be authenticated when the user logs into the Console. You do this by defining a `session-param` in the `weblogic.xml` deployment descriptor (see Listing 4).

In the final step, you need to package the Web application and deploy it to your WebLogic Admin server. You will need to copy the console extensions tag library descriptor file `$WL_HOME/server/lib/console_extensions_taglib.tld` to the `WEB-INF` directory of your web application.

While the space available here prohibits us from going into more detail about all of the nuances of creating Console extensions, we hope that the downloadable example will provide a realistic enough example to get you started. For more information on creating console extensions, please refer to the BEA WebLogic Server 8.1 documentation at http://edocs.bea.com/wls/docs81/console_ext/.

Summary

We've reached the end of our series on application management with BEA WebLogic Server 8.1. We hope that we were able to show you the power and flexibility of the WebLogic Server 8.1 management tools and services. The JMX infrastructure built into WebLogic Server not only allows you to create custom management applications to manage WebLogic Server, but also allow you to instrument your applications with JMX and expose this instrumentation through extensions to the WebLogic Console. 

Listing 1

```
InitialContext ctx = new InitialContext();
MBeanServer mbeanServer = (MBeanServer)
    ctx.lookup("weblogic.management.server");
ObjectName mbeanName = new ObjectName("wldj:Name=" + name);
Logger myLogger = new Logger(name);
ObjectInstance mbeanInstance =
    mbeanServer.registerMBean(myLogger, mbeanName);
```

Listing 2

```
ObjectName mbeanName = new ObjectName("wldj:Name=" + name);
Logger myLogger = new Logger(name);
ModelMBean modelMBean = new RequiredModelMBean();
modelMBean.setModelMBeanInfo(myLogger.getModelMBeanInfo());
modelMBean.setManagedResource(myLogger, "ObjectReference");
ObjectInstance mbeanInstance =
    mbeanServer.registerMBean(modelMBean, mbeanName);
```

Listing 3

```
<wl:node label="Standard MBean Loggers"
  icon="/images/folder.gif" expanded="false"
  url="/standard_summary.jsp">
```

```
<%
  for (int i = 0; i < standardNames.length; i++) {
    String name = standardNames[i];
  %>
  <wl:node label="<%= name %%"
    icon="/images/bullet.gif"
    url="/standard_detail.jsp?Name=<%= name %%"
  />
<%
  }
%>
</wl:node>
```

Listing 4

```
<weblogic-web-app>
  <session-descriptor>
    <session-param>
      <param-name>CookieName</param-name>
      <param-value>ADMINCONSOLESESSION</param-value>
    </session-param>
  </session-descriptor>
</weblogic-web-app>
```


The World's Leading Java Resource!

JAVA DEVELOPER'S JOURNAL

Here's what you'll find in every issue of *JDJ*:

- Industry insights
- The latest software trends
- Technical expertise
- Career opportunities
- In-depth articles on Java technologies

Subscribe Today &
SAVE 30% Off
the annual cover price

ANNUAL COVER PRICE
~~\$71.68~~
YOU PAY
\$49.99
YOU SAVE
30% Off the annual cover price

Sign up **ONLINE** at
www.javadevelopersjournal.com

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Acsera	www.acsera.com/demo	408-245-1000	19
BEA Systems	http://dev2dev.bea.com/medulla	800-817-4BEA	2
Cyanea	www.cyanea.com/wldj/underpar.html	877-CYANEA8	23
H&W Computer	www.hwcs.com/wldj2.asp	800-338-6692	13
HP	http://devresource.hp.com/d2d.htm	800-752-0900	39
ISSJ	www.issjournal.com	888-303-5282	29
Intersperse	www.intersperse.com	626-535-2000	31, 52
IT Solutions Guide	www.sys-con.com	201-802-3021	27
Java Developer's Journal	www.sys-con.com	888-303-5282	24, 45
MX Developer's Journal	www.sys-con.com/mx/subscription.cfm	888-303-5282	35
NetIQ	www.netiq.com/solutions/web	408-856-3000	9
Pantero	www.pantero.com	781-890-2890	11
PowerBuilder Developer's Journal	www.sys-con.com/pbdj	888-303-5282	42
Quest Software	www.quest.com/weblogic	949-754-8000	51
ReportingEngines	www.reportingengines.com	888-884-8665	25
SandCherry	www.sandcherry.com	866-383-4500	15
SYS-CON Reprints	www.sys-con.com	201-802-3026	43
Web Services Journal	www.sys-con.com	888-303-5282	47
Wily Technology	www.wilytech.com	888-GET-WILY	5

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

LOOK WHAT'S COMING NEXT IN THE ISSUE

Handling Large Database Result Sets

The Value List Handler is a well-known design pattern. There are, however, many trade-offs to consider when implementing this pattern. We offer some practical tips to make the pattern work, especially in large-scale J2EE applications.

Create a Real-World Business Process Model for Order Management

Complex business process management (BPM) solutions involving workflow creation, access to enterprise resources, and real business tasks, including human workflow, can become unmanageable. The workflows can grow into hard-to-follow decision trees, the developers having difficulty transforming the requirements into actual code, and become overly complex applications. This article is the first in a series of tutorials.

WebLogic on the Mac

Mac OS X, namely version 10.3 and better known as Panther, is a great Java development environment. This article is not a tutorial, or on Mac OS X, but an overview of where these technologies meet to create something that is completely unique and unparalleled in the industry; a J2EE development environment on a UNIX desktop done right.

Portal Tips and Tricks

We'll look at issues/quirks with Page Flows, navigation, portlet caching, entitlements, and other issues that seem to be bugging developers. Then, we'll discuss five unanswered questions from the Portal newsgroup and provide answers.

Develop And Deploy Java Portlets With Ease

Java Portlets using WebLogic Workshop 8.1 SP2 and deploying the portlets on WebLogic Portal 8.1 SP2 will be illustrated based on JSR 168. The article explains essential concepts such as portal, desktop, and portlets, and describes in detail various portlets modes and window states.

wldj THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC PROFESSIONALS



TRANSACTION MANAGEMENT

Transactions: Driving You to Distraction?

CHOOSE THE TYPE 2 DRIVER TO HELP



BY PETER HOLDITCH

AUTHOR BIO

Peter Holditch joined BEA as a consultant in the Northern European Professional Services organization in September 1996. He now works as a presales architect in the UK. Peter has a degree in electronic and computer engineering from the University of Birmingham.

CONTACT...

peter.holditch@bea.com

REPRODUCED WITH PERMISSION FROM BEA SYSTEMS.

One definition of a commodity is something that you take for granted.

I'll bet there aren't many readers out there who wake in the morning and exclaim, "Thank goodness there's air in the room to breathe!" Likewise, computer users will seldom give thanks for their operating systems, a proclamation like "praise be to those at AT&T and BSD for giving me Unix!" would likely raise more eyebrows than nods around an average water cooler. Things that are taken so much for granted are ripe for methods such as open source as provisioning mechanisms – at the end of the day, if its interfaces are well known and an implementation works "well enough" and is supported, then who cares if it was written in a garage or a high-tech lab? That's one of the reasons why I think the application server has some way to go before it commoditizes to the point where open source provides a realistic alternative as an underpinning to a business critical deployment – the application server market is small, and the capabilities offered by the market leading app. Servers are too rich (and the standards are evolving too quickly) for that to happen just yet a while.

But I digress (although I'm not sure if it's possible to digress before you have actually started...). This month's article is about a piece of technology that exists somewhere in the twilight zone between commodity and valuable asset – the humble JDBC driver.

One of the main drivers for the widespread adoption of the J2EE platform was the existence of standards that provide a good degree of plug-and-playability in terms of the resources accessing and accessed by the platform. One such principal resource is the SQL database (whose

adoption in its own life cycle was fueled by the standard relational model for data access providing some swapability at a different layer of the architecture). The Java standard that makes SQL databases pluggable at this level is, of course, JDBC. At a brief glance, you may be wondering what I am going to write about next, apparently being on the verge of saying that the JDBC driver is yet another commodity, but actually, the truth is, especially in the context of XA transactions, all JDBC drivers were not created equal. My intent this month is to dig a bit deeper into this, and help answer two questions: What's the overhead of XA and what JDBC driver should I use to access my Oracle database in an XA transaction?

What's the Overhead of XA?

Quite a few years ago, when X/Open first created the XA standard, there were no commercial databases that supported it. Then came Oracle 7 (and the other RDBMS products of around the same vintage) and XA moved from nice theory to realistic deployment option and the distributed transaction was finally liberated from the mainframe to roam free across the world of open systems. In those days, it was not unreasonable advice to suggest that architects build all their services to access the database via XA. The performance overhead of XA was small (I dimly remember figures of 5% bandied about) and the 1pc optimization built into Tuxedo (and later brought forward into BEA WebLogic Server's transaction manager) meant that the 5% was all you paid, if you didn't need a distributed transaction for any particular use-case.

These days, the world has moved on, and this advice is in need of amendment – it turns out that with recent versions of Oracle, and Java in

the picture, the overhead of accessing the database via XA is considerably more than 5% – some benchmarks put the degradation in throughput caused by using XA as high as 15x in the worst case (and note, this is not the overhead of two-phase commit, simply overhead introduced by the DBMS engine as a result of using xa_begin and xa_commit to delineate transactions).

Clearly, the prudent advice to anyone architecting a J2EE system is, at the very least, to benchmark your application using the XA configuration you need for production to ensure that your exact setup will give the throughput you require from your application, and be prepared to segment your database accesses into those that can use database local transactions and those that require XA for synchronisation with resources external to the database (for example, once and once only guaranteed message delivery from transactional message queues).

What JDBC Driver Should I Use to Access Oracle?

The JDBC standard provides for four different “types” of JDBC drivers. The important ones for this discussion are type 2 (in which the database is accessed via JNI calls from the java layer to Oracle’s native OCI libraries) and type 4 (in which a pure Java driver implementation talks across a socket to a remote server, which accesses the database as a proxy for the client)


Looking at this from a “commodity component” perspective, the choice between the type 2 and type 4 drivers might be expected to be academic (with perhaps a slight preference for type 4, given that this doesn’t require installation of the database native components on your application server box). This is by

“All JDBC drivers were not created equal”

and large the case for non-XA database access; however, for XA access the story is somewhat different. In this case, the throughput of the type 2 drivers is somewhat better than that of their type 4 equivalents, but more pronounced is the effect on the database CPU – XA access via the type 4 driver causes dramatically increased CPU utilization on the database server relative to the type 2 equivalent.

So, What’s the Conclusion?

As with all performance-type considerations, the main conclusion has to be that you should performance test an exact replica of your intended production configuration in order to determine that its performance will meet your needs. Any benchmark or other comparison of that kind is inherently heavily influenced by the usage patterns of the application, and also any benchmark is a snapshot in time of the state of the system – patches or new releases of the driver code (or the database engine itself) could completely change (hopefully for the better!) the results of any comparison.

However, with the technology where it is today, it seems safe to conclude that you should try to stick to the type 2 driver for XA database access, unless you have a pretty compelling reason to do otherwise. 

WebServices

.NET J2EE XML JOURNAL

LEARN WEB SERVICES. GET A NEW JOB !

SUBSCRIBE TODAY TO THE WORLD'S LEADING WEB SERVICES RESOURCE

Only \$69.99 for 1 year (12 issues)*
* Newsstand price \$83.88 for 1 year

Subscribe online at
www.wsj2.com or
 call 888 303-5252

*Offer subject to change without notice

Get Up to Speed with the Fourth Wave in Software Development

- Real-World Web Services: XML's Killer App!
- How to Use SOAP in the Enterprise
- Demystifying ebXML for success
- Authentication, Authorization, and Auditing
- BPM - Business Process Management
- Latest Information on Evolving Standards
- Vital technology insights from the nation's leading Technologists
- Industry Case Studies and Success Stories
- Making the Most of .NET
- Web Services Security
- How to Develop and Market Your Web Services
- EAI and Application Integration Tips
- The Marketplace: Tools, Engines, and Servers
- Integrating XML in a Web Services Environment
- Wireless: Enable Your WAP Projects and Build Wireless Applications with Web Services!
- Real-World UDDI
- Swing-Compliant Web Services
- and much, much more!



SYS-CON MEDIA

SYS-CON Media, the world's leading /-technology publisher of developer magazines and journals, brings you the most comprehensive coverage of Web services.



ADMINISTRATION

The Promise of Utility Computing Today

DEPLOYING A SHARED WEBLOGIC INFRASTRUCTURE FOR J2EE APPLICATION HOSTING



BY TIM JACOBSON & TRACE LOWE

AUTHOR BIOS...

Tim Jacobson is the chief architect for the Shared Application Server Utility at HP. He is responsible for the strategic architectural direction of the program and integration of new technologies and applications into the infrastructure.

Trace Lowe is the J2EE Administration Technical Lead for SASU at HP.

CONTACT...

tim.jacobson@hp.com
trace.lowe@hp.com

Business managers are demanding better, faster, and cheaper access to IT resources and environments. At the same time, IT budgets and resources are being cut, there is a proliferation of servers running at low utilization.

These competing objectives play nicely into consolidation and virtualization of computing resources. A strong case can be made for deploying a virtualized and shared environment designed for hosting multiple BEA WebLogic applications with a high level of isolation between each application.

The Current Problem

Today IT organizations face the challenge of consistently and strategically aligning IT investments with evolving technology and changing business objectives. At the same time, IT must optimize assets, reduce complexity, be cost effective, comply with standards, and improve the stability and flexibility of the environment. Within HP the J2EE application server platform, specifically BEA WebLogic Server, was identified as an area where applying concepts and solutions from utility computing could bring significant benefits to IT groups and the business units served.

In HP's case, there wasn't a federated approach for managing J2EE environments across business units, creating challenges in assuring a cost effective platform. For example, HP internal business units invested in additional hardware, software, and support resources to host their application runtime environments on a per application basis. This led to three significant problems:

1. Redundant infrastructure and support resources across the organization, leading to excess spending and often low resource utilization of each infrastructure
2. Increased time-to-market for solutions because of the time and budget required to

3. Lack of standardization, repeatable processes, and solutions across application teams.

These multidimensional problems impact application teams by reducing their agility and increasing their expense when delivering solutions. Meeting these challenges allows IT to manage the environment better, faster, cheaper, simpler, and smarter.

Our Solution

In early 2003 a team was built to focus on this problem and began a strategic program to develop a Shared Application Server Utility (SASU) infrastructure. This team was chartered to create an adaptable, flexible, and fully monitored environment for hosting J2EE applications on shared virtualized hardware. The goal was to provide a J2EE application hosting service that business partners can use on-demand and eventually be charged back based on actual environment usage. SASU provides the infrastructure and the application provider provides the application and team resources for deployment and support. The infrastructure now runs in production with several applications and has a funnel with many new pending requests.

To deliver the infrastructure, SASU integrates a set of complex emerging technologies to deliver a comprehensive solution. Its servers are deployed in a data center that provides Web server farms, load balancing appliances, external access capabilities, Storage Array Network, and shared database farms. Figure 1 shows the overall SASU infrastructure deployed on a physical machine. The infrastructure can be described in terms of layers.

J2EE Environment Layer

BEA WebLogic Server 8.1 SP2 provides the J2EE platform, including advanced capabilities such as isolated application environment sup-

port, administration services, clustering across physical machines, and session state management.

SASU has deployed dedicated domains per application, including dedicated managed servers (often deployed to multiple machines) and a dedicated admin server. This will provide JVM isolation and the ability to deploy, and start and stop applications independently.

Monitoring and Reporting Layer

SASU has deployed monitoring and reporting components based on the HP OpenView product suite. This provides application alerts, historical usage reporting per application, service-level objective violations, and the ability to deploy application specific monitoring requirements through ARM or the inclusion of upcoming components to provide URL availability and transactional level information. The information is captured into a central reporting database that allows for the creation of reports, or for additional automation around responding to specific events occurring in the system.

Management and Control Layer

This is a critical component for providing isolation of shared CPU resources between applications. The key technology is an HP-UX software product - Work Load Manager (WLM).

WLM provides a configuration file where CPU Min/Max resources can be allocated and guaranteed for a specific application process through the definition of a WLM workgroup with an associated service-level objective (SLO). The granularity for the allocation of CPU can be up to 1% of 1 CPU for a workgroup. WLM periodically monitors applications performance characteris-

tics versus the configured service-level objectives of the configured applications. When additional CPU is required to meet increasing load, WLM automatically allocates the additional CPU to keep the application within the min/max range up to its max value, at which time the CPU allocation is capped.

A key benefit of this technology is that system utilization can be pushed to higher levels as applications can get access to the reserve CPUs when required but share the excess capacity with others when not utilized. In addition, errant applications will be capped at their configured max configuration, which will stop an application from overtaking all of the CPU reserve and impacting other applications.

Physical Disk Layer

SASU deploys application disk space into a Storage Area Network with dedicated logical volumes per application that are sized according to an application's needs up front. This allows a specific application's disk space to be completely isolated from others on the system. All application-specific files and logs will reside in this space.

Processes and Automation

SASU provides a consistent and repeatable development and promotion to production methodology, including standard shared functional test, integration test, and production infrastructure and maintenance. This enables application development teams to focus on their application functionality without dealing with infrastructure configuration and maintenance, enabling simplified and accelerated application deployment, and testing and repeatable environment creation steps. An essential component for delivering environments quickly and in a repeatable manner is the automation of the WebLogic domain creation.

Automating WebLogic Domain Creation and Configuration

Life Before Automation

Imagine you're hosting multiple applications in a shared WebLogic environment. Two to three new applications must be deployed each week, each requiring its own dedicated domain with users, passwords, port numbers, JVM arguments, classpath configuration, and a host of other unique server properties. Creating these by hand, using the Domain Configuration Wizard, administration console,

and other tools, can easily become a full-time job.

Automation Done Thus Far

Wanting to get out of the "hands-on administrator" business, we embraced "wshell" to assist our automation effort. We began by creating a flat configuration file. The file contains wshell environment variables for all the static and dynamic configuration variables which make up the common MBean properties needing configuration in a new domain. This configuration file is used as input to a domain configuration wizard script template, which creates a basic entry-level domain. After the basic domain is created, we start the admin server and run generic wshell scripts that set all the properties defined in the domain environment file. This automation has exponentially shortened the amount of time and effort necessary to create an application domain.


What Lies Ahead

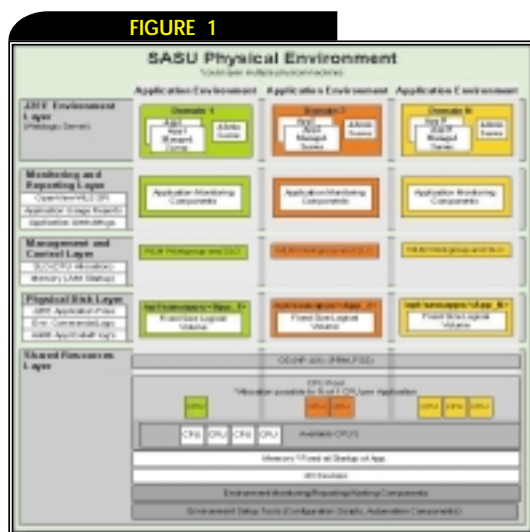
Investigation is now under way automating other areas, including uploading and tracking application versions, as well as deployment. Going forward, we must be careful not to "over-automate" in order to ensure that the interface and functionality remain flexible. The key is in striking the right balance between the GUI approach, and automation.

Another key automation candidate is the "promotion" of applications between environments (for example, taking an application from test to production). BEA is working on features such as deployment plans in future releases, which should ease configuration issues when promoting applications. Any automation work must account for and take full advantage of such features.

Automation not only increases the speed and efficiency of operations, but also provides a highly "repeatable" process that is much less error prone than manual processes.

Things to Consider

Embracing a shift from traditional infrastructure to utility computing requires changes across three dimensions: people, processes, and technology. It is critical to define a solution that takes all three dimensions into account. A strong technical solution and architecture will not be successful without a buy-in from application teams and sponsors for the changes in process to be part of a shared environment. With a solid plan and organization structure both application teams and IT sponsors should benefit from reduced costs and faster time-to-market offered from a shared infrastructure. 



SASU infrastructure

They will tackle regulatory issues such as the Patriot Act and Sarbanes-Oxley, accounting best practices, complex and politically charged interorganizational reference data issues, risk management, and data caching policies and SLAs for grid computing, to name a few.

Consider the reference data issue: a Tower reference data report tells us (1) organizations are spending \$3.2M for reference data annually; (2) 48% of interviewed organizations revealed that reference data are contained in more than 10 systems, and 8% stated a staggering 150 systems or more; (3) poor quality of reference data causes 30% of business failures; and (4) the same Tower report reveals that manual entry and error-prone manual maintenance is still a widespread practice. Need I continue? You get the idea. Traditional EAI techniques, data replication, and naive notions of enterprise-level data normalizations not only had limited success but actually magnified the problem by cloning bad, unreliable, and conflicting data throughout the enterprise.

But how can a BPMS-based solution solve such a gargantuan problem? The complete answer and strategic approach can fill a chapter or two of a practical guide to BPM, or be a multimillion project in its own right. Without going into many details, here is the approach: visualize a piece of reference data having a number of attributes associated with it, say an order of magnitude of 600 or so. The implied assumption is that different organizations within an enterprise have first authoring rights into different segments or clusters of attributes; therefore, segregate the attributes into principal clusters by primary LOB user owners, say six domains, each one having 100 attributes or so. Finally, for each domain design and implement processes that manage the life cycle of each subset of attributes and include policies about the cluster's life cycle and approvals (see Figure 3).

In other words take the departmental ownership rights, procedures, and policies and implement them in BPMS. That's what BPM is all about, right? Executable processes. The heart of the problem is now the key to a successful solution. And last but not least, you have solved yet another challenging issue around enterprise integration: corporate governance and ownership of the technology solution.

The academic community has developed a set of 20 basic patterns that accurately define all the major workflow patterns. Workflow IDEs can then be measured against the established patterns. These are divided into six broad categories: (1) basic control flow, (2) structural, (3) state based, (4) advanced branching and synchronization patterns, (5) cancellation patterns, and (6) multiple instances.

1. **Basic control flow patterns are** (a) sequence, (b) parallel split, (c) synchronization, (d) exclusive choice, and (e) simple merge.
2. **Structural patterns are** (a) arbitrary cycles and (b) implicit termination.
3. **State based patterns are** (a) deferred choice, (b) interleaved parallel routing, and (c) milestone.
4. **Advanced branching patterns are** (a) multi-choice, (b) synchronization merge, (c) multi-merge, and (d) discriminator.
5. **Cancellation patterns are** (a) cancel activity and (b) cancel case.
6. **Multiple instances patterns are** (a) multiple instances without synchronization, (b) multiple instances with a priori design time knowledge, (c) multiple instances with a priori runtime knowledge, (d) multiple instances with a priori design time knowledge, (e) multiple instances without a priori design time knowledge.

Wil van der Aalst of Technical University of Eindhoven University provides a detailed account of some of the patterns above through an example process.

Patterns (3)b, 5(a) and (b), all of (6) require messaging and Web services patterns. These patterns can be divided in the following categories: service access and configuration patterns including wrapper façade, component configurator, and interceptor. Event handling patterns, including reactor, proactor, and acceptor-conductor and concurrency patterns, including active object, monitor object, and leader/followers architectural patterns. Douglas Schmidt, et al, in *Pattern-oriented Software Architecture (Vol 2): Patterns for Concurrent and Networked Objects* provide an excellent account of the above patterns. The Addison Wesley Signature Series from Martin Fowler, et al, is also an other great source for connectivity patterns.

The BEA 8.1 WebLogic Platform takes away much of the need it for low-level implementation. However, it is important to realize that since BPM is indeed a programming paradigm in its own right, the prospect of creating spaghetti processes is as real as BASIC spaghetti full of GoTos. Proper design is highly recommended!

Summary

In this article I described the reasoning behind modeling, the need for modeling the model, and the state of the current state in the standards front for BPM. I described how BPMS provides a top-down incremental approach unifying all the knowledge domains within the enterprise and presented a high-level best practice approach to the daunting problem of enterprise reference data. Finally, I presented a survey of workflow and connectivity patterns.

In the final article of this series I'll describe the get insurance quote business case and present BMP modeling options. I will then implement a solution with BEA's 8.1 WebLogic Platform using some of the workflow and connectivity patterns I presented in this article and then discuss some of the limitations that still exist and possible solutions.

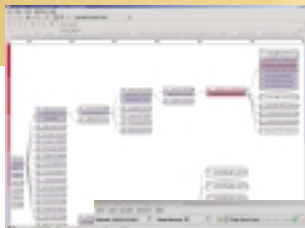
Until then: processes are everywhere. Can you see them?

References

- Engberg, U. and Nielsen, M. (1986). "A calculus of communicating systems with label-passing." Report DAIMI PB-208, Computer Science Department, University of Aarhus, Denmark.
- Milner, R.; Parrow, J.; and Walker, D. (1992). "A calculus of mobile processes, Parts I and II". *Information and Computation*, 100, 1, pp 1-77.
- van der Aalst, W. "Classical Petri nets: The basic model." <http://tmitwww.tm.tue.nl/staff/wvdaalst/Courses/pm/pm2classicalpn.pdf>
- ter Hofstede, A. (QUT); Kiepuszewski, B. (QUT); Barros, A. (UQ); Ommert, O.(EUT); Pijpers, T. (ATOS); et al. *Workflow patterns*. www.tm.tue.nl/it/research/patterns/
- Booch, G.; Rumbaugh, J.; and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.
- Hohpe, G. and Woolf, B. (2004). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.
- BPEL4WS, Business Process Execution Language for Web Services. BEA, IBM, Microsoft.
- BPML, (2002). *Business Process Modeling Language*. Bpmi.org.
- Reference Data, (2001) "The Key to Quality STP and T+1." A Tower Reuters CAPCO report.
- Graham, I. (2000). *Object-Oriented Methods: Principles and Practice*. Addison-Wesley. 🍎



SPEND LESS TIME PROBLEM SOLVING... AND MORE TIME DEVELOPING APPLICATIONS.



PerformaSure – a system-wide performance diagnostic tool for multi-tiered J2EE applications running in test or production environments.



JProbe – a performance tuning toolkit for Java developers.

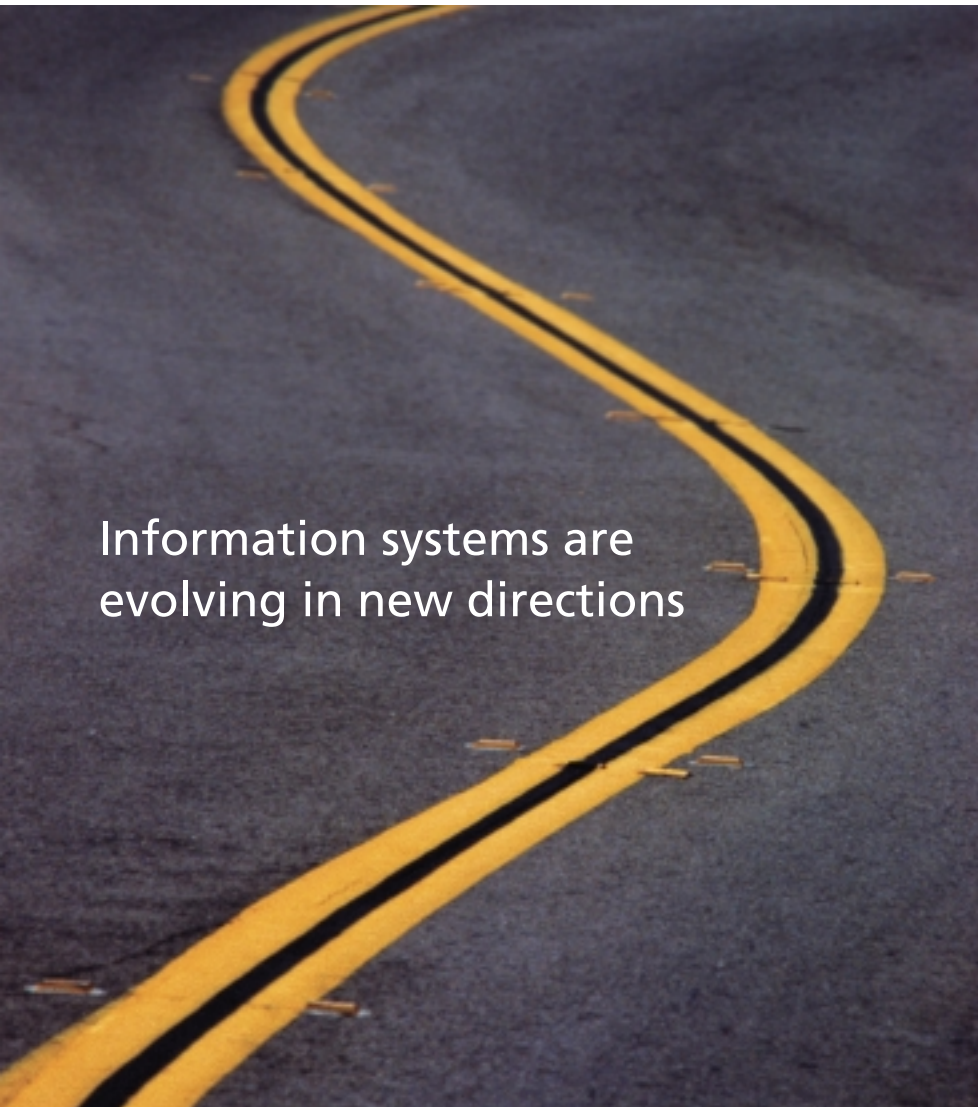
Join The Thousands of Companies Improving Java Application Performance with Quest Software.

Whether it's a memory leak or other performance issues, Quest Software's award-winning Java products – including JProbe® and PerformaSure™ – help you spend less time troubleshooting and more time on the things that matter. Quest's Java tools will identify and diagnose a problem all the way down to the line of code, so you no longer have to waste time pointing fingers or guessing where the problem lies. Maximize your team's productivity with Quest Software by downloading a free eval today from <http://www.quest.com/weblogic>.



© 2004 Quest Software Inc., Irvine, CA 92618 Tel: 949.754.8000 Fax: 949.754.8999

Eliminate the Risk of SOA Deployment



Information systems are evolving in new directions

With enterprise information systems becoming more vital to business success...

With J2EE and SOA changing the face of your enterprise...

With distributed, interconnected systems becoming the norm...

How will you manage?

With Intersperse, that's how.

Intersperse Manager is the only product designed specifically for production management of SOA environments. Intersperse Manager lets you discover, monitor, and manage the J2EE-based SOA applications that run your business.



Management solutions
for service-oriented
enterprise applications

626.535.2000
www.intersperse.com